Guide to Operations

PANOS

USER GUIDE

Guide to Operations

PANOS

ACORN SCIENTIFIC

PART NO 0410, 011
ISSUE NO 1
JULY 1985

The product described in this manual and products for use with it, are subject to continuous developments and improvement. All information of a technical nature and particulars of the product and its use (including the information in this manual) are given by Acorn Computers in good faith.

In case of difficulty please contact your supplier. Deficiencies in software and documentation should be notified in writing, using the Acorn Scientific Fault Report Form to the following address:

Sales Department
Scientific Division
Acorn Computers Ltd
Fulbourn Road
Cherry Hinton
Cambridge
C131 4JN

All maintenance and service on the product must be carried out by Acorn Computers' authorised agents. Acorn Computers can accept no liability whatsoever for any loss or damage caused by service or maintenance by unauthorised personnel. This manual is intended only to assist the reader in the use of the product, and therefore Acorn Computers shall not be liable for any loss or damage whatsoever arising from the use of any information or particulars in, or any error or omission in, this manual, or any incorrect use of the product.

Published by Acorn Computers Limited,
Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN.

Within this publication the term BBC is used as an abbreviation for the British Broadcasting Corporation.

NOTE: A User Registration Card is supplied with the hardware. It is in your interest to complete and return the card. Please notify Acorn Scientific at the above address if this card is missing.

ISBN 0 907876 43 9 Acorn Scientific

Contents

1 Introduction

1.1 Context

This document describes the Panos user interface. As its name suggests, it is a guide to operating Panos from the keyboard - how to edit files, how to list directories etc. It does not deal with installation; this is described in the *User Guide* supplied with the system.

More specialised material, which will be of use to software developers, can be found in the companion *Panos Programmer's Reference Manual* which is primarily concerned with the Panos procedure library.

More specifically, the items dealt with in this Guide are as follows:

- the concepts (or user model) that govern the behaviour of Panos
- the organisation and configuration of a Panos system
- the command language for requesting Panos to carry out an action
- the operation of the Panos utilities including the editor and linker.

Panos provides the base for all systems software supplied with Cambridge Series hardware with the exception of BBC Basic.

1.2 Panos Design Objectives

Panos was designed with the following objectives:

- to provide minicomputer user facilities
- to provide microcomputer control over the environment
- to be a specialist single user system
- to be installable and maintainable by the end-user
- to provide support for professionals: both for end-users and programmers
- to be relatively simple, yet extensible
- to be capable of operating on modest configurations
- to take advantage of fast, 32 bit computer power

- to integrate with the BBC Microcomputer hardware and software
- to exploit networking and other communications
- to support high level language programming in many languages
- to permit ease of learning and use

1.3 Conventions in this Manual

The following conventions are observed in this publication:

1. Numbers not in decimal are prefixed by their base, for example 16_1A is decimal 26; -2_1010 is -10 in decimal.
2. Angled brackets refer to a class of objects, for example < device name > means any one device.
3. Square brackets or braces enclose optional items.
4. The term `BBC Microcomputer' should be extended to include the term '10 Processor' as used in the Cambridge Workstation, and vice-versa.

1.4 Organisation of this Manual

This manual has been carefully structured to permit its use both as a reference guide, and as a primer. This has been achieved by dividing it into many relatively self-contained sections and sub-sections, ordering these sections for the sequential reader, but providing numerous cross-references for the browser. Some sections contain more specialist material; these may be omitted on first reading.

The manual and its constituent parts have been separated into:

- concepts that define the `user model'
- organisational and configuration details
- principles of the command language
- feedback to the user (including errors)
- operational (`how to') details.

This structure applies both to the manual as a whole, and to individual chapters where applicable. Thus Chapters 1 to 5 introduce, then describe general concepts, organisation, command language, and feedback for Panos as a whole. Chapters 7,8 and 6 detail operational use of the editor, linker, and remaining utilities respectively. These three chapters are structured to some extent in the same way as the whole, i.e. describe particular concepts, command languages etc.

Before embarking on the operational details it will be helpful to read at least some of the preliminary material.

2 Concepts

2.1 Introduction

Context

This chapter presents many of the concepts that determine the `user model' of Panos. They are described in a manner suitable for the end-user rather than for the programmer.

Many of these concepts have an equivalent form for the programmer, and as such are described in the *Panos Programmer's Reference Manual* as follows:

Storage Management	Module Store	Chapter 5
Input and Output	Module IO	Chapter 6
Filing Systems	Module File	Chapter 7
Program Loading	Module Loader	Chapter 8
Condition Handling	Module Handler	Chapter 11
Asynchronous Events	Module Handler	Chapter 12
Global Strings	Module GlobalString	Chapter 13
Program Control	`Module Proarnm`	`Chnnfpr 14`

Organisation of this Chapter

This chapter simply describes in turn the conceptual models for the user interface, program control, input and output, filing system, event handling, global variables, and configuration data.

2.2 User Interface Model

The model for interaction with the user is described in this section.

Input/Output, and hence dialogue with the user is based on the notion of streams. There are four such logical streams associated with Panos:

input stream
output stream
control stream
error stream

The input stream is used for the data being operated on by a program, for example numeric values by a user program, or text prepared for a text formatter.

The output stream is used for the results of a program, for example a table of values from a user program, or formatted text from a text formatter.

The control stream is for input from the user to control the action of a program, for example the response to a prompt, or the name of a program or command to run. Associated with the device for such input is the device used for the output of prompts. By default the control stream uses the screen (vdu) for output, and the keyboard for input.

The error stream is for feedback to the user, i.e. error messages, warnings, or for confirmatory messages to provide reassurance. By default the error stream uses the screen.

Logical streams may be associated with physical streams and hence with actual devices in a number of ways. If the user takes no specific action, the default mappings as indicated above are used. Alternatively, the user may explicitly redirect input or output to a particular device. For example, error messages could be redirected to a printer.

There is no requirement on programs to use logical streams, so many will use direct I/O from or to physical streams instead. However, where appropriate, all the supplied systems software follows the model of interaction as described. Error and Control streams are used by most system programs, input and output streams are however mainly applicable to filter type programs.

Interaction with the user is also governed to some extent by the values of certain system attributes:

- global (environment) variables
- vdu characteristics
- tab settings
- function key bindings
- date and time
- working directories

These attributes have certain default values or may be set by users for the duration of Panos use. They are described in full later.

2.3 Program Control Model

Panos provides support for the execution of programs, both user and system. More specifically it provides a runtime environment for programs, in particular for programs written in high-level languages, and including those written in mixed languages.

Panos supports a procedural model of program execution. This means that programs may call other programs and resume execution on return. This property is capitalised on by the command line interpreter and by the editor.

In addition to the execution of single programs, Panos is also able to obey command sequences, and thus to execute a sequence of programs. This is achieved through the use of command files.

Communication between two programs running sequentially or procedurally may be achieved through the use of global variables.

When a program is invoked from its parent (which typically would be the command line interpreter), its initial environment consists of:

- the logical streams of its parent (see 2.4),
- its own memory,
- the same environment for events as its parent (see 2.6)
- the same global variables as its parent (see 2.7), since these are a shared resource.

For full details see the *Panos Programmer's Reference Manual*

2.4 I/O System Model

Panos presents its own device and filing system models to the user. These models are currently implemented by lower level BBC Microcomputer mechanisms. For example, Panos files are simply stored and maintained by a filing system such as the DFS (for floppy disc), ADFS (for Winchester or MFM floppy), or NFS (for file server).

The user need not be aware of the lower level implementation except perhaps when transferring BBC Microcomputer files to or from Panos, and to some extent when the underlying filing system imposes certain restrictions.

Panos is able to provide more useful higher level properties than the raw filing system, for example, time-stamping of files, file name extensions, and filing system names forming part of a file specification.

Within Panos, devices and filing systems are treated uniformly wherever possible. For instance, when using the utilities, specifying a device or filing system for output follows the same format.

In this section the Panos I/O model is described. This applies to all devices. The Panos filing system model for devices that maintain directories extends the I/O model, and is described in the next section.

For further details, refer to Chapter 6 of the *Panos Programmer's Reference Manual.*

2.4.1 Streams

Input/output is based upon the concept of streams. An I/O object may be a device or a file, and must be identified by a string (i.e. textual name) with one of the following syntactic forms:

(a) < device name >
(b) < filing system name > : < file name >
(c) < file name >

The case (upper or lower) of < device name > and < filing system name > is not significant. The actual names of currently supported filing systems and devices are listed in Tables 2-1 and 2-2.

Examples (most showing a file called data-dat) are:

(a)	**vdu:**	**screen (visual display unit)**
(b)	**adfs:$.data-dat**	**file in root directory on adfs**
(c)	**data-dat**	**file in current working directory**
(d)	**dfs:data-dat**	**file on current DFS drive**
(e)	**dfs::2.data-dat**	**file on DFS drive 2**

Note the position of the colon after the filing system or device name. Because the DFS filing system also uses a colon to specify the drive number, in some cases, two colons will need to be used; one for the device, and one for the drive number, as in example (e).

The streams described in this section are physical and refer to actual devices; the logical streams of the Panos model of interaction may be mapped onto physical streams, (see 2.2 and 4.9.1). The logical streams are listed in Table 2-3.

Table 2-I Filing System Names

DFS	- disc filing system (floppy disc)
ADFS	**- advanced disc filing system (Winchester, MFM floppies)**
NFS	**- network filing system (for Econet file server)**

Table 2-2 Physical Devices

vdu:

Refers to the screen (output only) with filtering of control characters. Only ASCII characters (32..126), clear-screen (FF), newline (NL = LF) and carriage-return (CR) are sent to the screen. All others are output as a pair of hexadecimal digits enclosed in square brackets.

rawvdu:

Refers to the screen (for output only) with no filtering. The effect is exactly as defined for VDU codes.

kb:

Refers to the machine's keyboard (input only) with both carriage-return (CR) and line-feed (LF) being read as newline (NL). Input is buffered and line editing (see 4.4) is enabled. Only the printing

characters and line-editing characters are accepted, plus ESCAPE and CTRL - D. All others are ignored.

rawkb:
Refers to the keyboard (input only) with no translation or filtering of characters. Raw characters are read directly from the keyboard and are not echoed.

bbc:
A combination of rawvdu: for output and rawkb: for input.

tt:
A combination of vdu: for output and kb: for input.

RS423:
Refers to the serial line (input or output).

printer:
(or lp:) Refers to the printer (output only).

null:
Refers to a `sink'. Output to this device is discarded. Input from this device appears as if end of file were reached immediately.

Table 2-3 Logical Streams (Special Devices)

Input:	Current Input Stream
Output.	Current Output Stream
Control:	Current Control Stream
Error:	Current Error Stream

2.5 Filing System Model

The Panos filing system model presents a logical filing system to the user. This model is currently implemented partly by Panos directly, and partly be mappings onto more primitive BBC Microcomputer, i.e. physical filing systems. In principle, the user need only know about the Panos model, but see below.

Wherever possible Panos presents a uniform view regardless of the physical filing system, but where there are differences, limitations of the physical system may impose restrictions. There are three possibilities.

Firstly (and usually) the user request is carried out exactly as requested. Secondly, Panos will sometimes be able to make an obvious interpretation, for example, date-stamps have different granularity in different systems, so fractions of a second may need to be rounded. Finally, sometimes there will be restrictions, for example a legal length Panos name will always map onto an ADFS or NFS name, but not onto a DFS name. The user request will be rejected in such a situation.

The Panos filing system model has the following components which are described in detail in the remainder of this section:

- file names
- file name extensions
-time and date stamping
- access rights (permissions)
- hierachical directory structure

Together with size, these determine the file attributes.

2.5.1 File Names

Panos basic file names (with no directory or drive prefix) consist of a base filename plus an extension separated by a hyphen ..... Names may be of any length, although in practice the physical filing system will impose constraints.

Legal characters in names are:

alphanumeric characters	A-Z, a-z, 0-9,
special characters	!_/

The characters $, &, _, , . : and - have special meanings (see below). File names are case (lower/upper) insensitive, that is case is ignored in referring to file names, although files are stored with names in mixed case.

2.5.2 File Name Extensions

The Panos filing system supports the concept of typed files through the mechanism of file name extensions. Those defined by Acorn are shown in Table 2-4. File typing is not enforced, but many of the system programs such as the language compilers rely on these conventions.

File name extensions are from 0-4 characters in length, plus the prefix character "='. Since they form part of the file name, they consist of the same legal characters.

Extensions are mapped onto the physical filing system. At present these use rules defined by the user, but set up in the !Panos initialisation file through the mechanism of global variables.

The mapping is from the Panos file extension onto a filing system directory. On the ADFS and NFS, the directory takes the same letters as its name prefixed by an underscore. On the DFS, the directory has a single-letter name as shown in Table 2-4.

So, for example, the file name `Prog1-Pas' will map onto a directory called `_pas' on the ADFS or NFS, and its physical name is `_pas.Prog1'. On DFS, the extension `-Pas' is mapped onto a single-letter directory, `p', and its physical name is thus `p.Prog1'. This is achieved (in !Panos) as follows:

```
-> set var file$dfs:-pas "p.-"
```

Although it is normally necessary for the user to create directories explicitly on the ADFS and NFS for the purpose of file store organisation, it is not required in this context. Such directories are `hidden' within Panos, although they will become explicit when, say, cataloguing a Winchester disc under Pandora or on a BBC Microcomputer. In fact, the user normally need only be aware of these mappings when reading or writing BBC Microcomputer files.

Standard Conventions

By convention, certain file name extensions and associated meanings are employed within Panos. These are shown in Table 2-4 along with the DFS mapping. Table 2-5 shows those DFS directories reserved for use by Acorn. and those available for users or for packages.

Table 2-4 File Types and Extensions

extn	DFS	Meaning
-abs	x.	Absolute binary file (for execution under Pandora).
-aof	o.	Object file in Acorn Object Format.
-asm	a.	Assembler source file.
-bbc	6.	BBC Microcomputer host code.
-c	c.	C source file.
-cmd	e.	Command file.
-dat	d.	Data file (arbitrary format).
-f77	f.	Fortran77 source file.
-h	h.	header file for use with C.
-lib	q.	Object file in AOF used as a library.
-lis	y.	Listing file.
-lsp	l.	Lisp source file.
-map	m.	Map files (FORTRAN 77 and Linker)
-pas	p.	Pascal source file.
-rif	r.	Relocatable Image produced by Panos Linker.
-src	s.	Source text (program in unspecified language)
-tmp	z.	Temporary file.
-txt	t.	Text file.

Also, for release on floppy disc only

$.	Pandora files.
i.	Lisp image directory.

Table 2-5 Unused DFS Directories

Reserved for Acorn

j.	n.	
7.	8.	9.

Available for Use

b.	**g.**	**k.**
u.	**v.**	**w.**
0.	**1.**	**2.**
3.	**4.**	**5.**

2.5.3 Time and DateStamping

Panos files may have associated with them the time and date that they were first created. In practice many of the utilities such as the editor `create' the file afresh every time it is updated. If the time has not been set by the user when switching the machine on or typing** `CTRL – BRE K` **then the date stamp will be `unset'. Non-Panos files such as Basic programs appear to Panos as if they too have unset date stamps.

The date stamp is currently stored as part of the directory entry in the fields normally used by the load and execution addresses (which Patios does not need as such). Care is therefore required when using Panos to copy non-Panos files. Use the -exact option with the Copy command.

2.5.4 Access Rights (Permissions)

The Panos model of permissions (rights to access a particular file) follows that of the ADFS or NFS. For details, see for example the *Winchester Disc Filing System User Guide.* **See also the Access command.**

2.5.5 Filing System Structure

The Panos model of filing system structure follows that of the ADFS or NFS, i.e. it is hierarchical. For full details, see for example the *Winchester Disc Filing System User Guide.*

The ADFS and NFS have a hierarchy that can extend to any depth, i.e. at any level the contents of a directory may be `leaf' files or further directories. The top level directory is called the `root'. On the NFS each user is allocated a directory further down the structure called the 'log-on' directory.

Although the DFS has a flat directory structure, Panos treats it in the same way subject to the constraint that there is only one level of sub-directory possible below the root directory, so the longest path on a given drive is $.<dir>.<name >.

Associated concepts are directories, pathnames, object specifications, and working directories; these are now described in turn.

Directories

Panos currently inherits the properties of the physical filing systems, thus, for example, there is a limit of 47 separate files per directory within ADFS. There are no separate directories within Panos on the DFS since the single level structure provided is used by Panos for the file name extension.

Object Specifications and Pathnames

To refer to a file, i.e. **to give an `object specification', it is always possible to supply a full `Pathname'. For example:**

adfs::0. $.Reports.May. Visit-txt
dfs:: l . $. Visit-txt

The first might be a text file describing a visit residing in directory May which itself resides in directory Reports which is in the root directory of the ADFS filing system on a Winchester. The second might be the same file but on drive 1 of a floppy disc.

The full pathname starts with a device and/or drive name, and is followed by a file name starting with $ or &.

In practice file references tend to be localised, so there are a number of mechanisms for shortening the full pathname. These include the use of `(current) working directories', of special symbols, and of wild symbols.

Working Directories

Associated with each filing system on a particular machine is a `working directory'. One of these is the `current working directory'. Initially the working directory is the root directory for each device on drive number 0, although !Panos as supplied will alter this. The current working directory

may be changed by means of the Set command, and inspected with the Show command. This also will change the working directory for the filing system referenced, and this will remain in force until a subsequent change for that filing system.

A relative object specification is one for which the file name does not start with $ or &. Such a specification is taken to refer to the working directory. If a device name is supplied, then it refers to the working directory for that device, otherwise to the current working directory.

A full path name has $ or & starting the file name. Such a specification is taken to refer to the root or log-on directory respectively. If a device name is supplied, then it refers to that device, otherwise to the device associated with the current working directory.

For example, if there were a command called `select', the following series of commands would refer to files whose full path names are shown on the right hand side:

```
-> select $.index-txt             adfs::0.$.index-txt
-> select $.heroes.ludwig-txt adfs::0.$.heroesLudwig-txt
-> set dir $.heroes
-> select ludwig-txt              adfs::0.$.heroes.ludwig-txt
-> set dir dfs::1
-> select johann-txt              dfs::1.$.johann-txt
-> select adfs:ludwig-txt     adfs::0.$.heroes.ludwig-txt
```

There is a further, and separate mechanism for referring to the name of a program or command file to be executed. See under `Search Path' in section 4.2.2.

The special symbols that may form part of an object specification are shown in Table 2-6:

Table 2-6 Special Svmbols

$	root directory
&	log on directory
®	(current) working directory
	parent directory
-	file name extension separator
	directory separator

2.5.6 Examples

Examples (for the file data-dat) are:

(a) data-dat	**in current working directory**
(b) adfs::0.$.data-dat	**full path name**
(c) adfs:data-dat	**in adfs working directory**
(d) MyDir.data-dat	**in directory MyDir within current working directory**
(e) $.MyDir.data-dat	**in directory $.MyDir**
(f) data-dat	**in parent directory**
(g) dfs::2.data-dat	on floppy disc drive 2
(h) dfs:d.data	**physical file name**

2.6 Event Handling Model

Panos provides facilities for dealing with both synchronous and asynchronous events. In this context, the former are sometimes called conditions or exceptions.

Exceptions are synchronous to the flow of program execution, and may be `hard' e.g. division by zero, or `soft', i.e. signalled by the programmer through his or her code.

Asynchronous events are generated by interrupts in the I/O Processor, and include the ESCAPE key being pressed.

Events may be used within a program to help organise control flow. Often they are used to terminate a program when an `error' occurs. See Chapter 5 for more details. **It is possible for the high-level language programmer to trap events and take suitable recovery action. See the** *Panos Programmer's Reference Manual,* **Chapters 11 and 12 for details.**

2.7 Global Environment Variables

The way in which Panos behaves is determined to an extent by the settings of the global variables sometimes referred to as global strings. See 2.3 for the context of their use, and Chapter 13 of the *Panos Programmer's Reference Manual* for a full description.

A global variable has a name and a string value. To alter the value of a variable use the Set command, (or the Set built-in command). To inspect the value of one or more variables use the Show command. The SYS$ variables are an exception: they cannot be changed in this way.

To set the value of a variable each time Panos is run, edit the !Panos command file.

For example, to change the Panos prompt, then inspect the values of the PROGRAM$ variables, type:

```
-> set var CLI$prompt
& show var PROGRAM$*
```

Enclosing the value of the global variable in double quotes is not normally necessary, but will allow leading and trailing spaces to be included as in the example.

A variable is declared through being set, i.e. before it is given a value it does not exist. Some variables, e.g. CLI$Prompt are initially set by the system.

2.7.1 System Defined Variables

There are a number of global variables defined by the system, most of which are initialised by Panos itself, or whenever Panos is entered through execution of the !Panos command file. These variables generally hold information relating to the state of the environment, hence their name.

The global variables defined by the system are shown for completeness in Table 2-7. The meaning and significance of each variable or group of variables are explained in the sections referred to in the Table.

Table 2-7 System Defined Global Variables

CLI$ResultCode	result code of last program to run	5.4
CLI$Path	search path for Panos commands	4.2.2
CLI$Prompt	Panos CLI prompt	4.2.3
CLI$Echo	determines echoing of command files	4.8.1
CLI$Stop	defines termination condition in command files	4.8.1
FILE$-ext	file transformation for `ext'	2.5.2
FILE$dfs:-ext	file transformation for `ext' on DFS	2.5.2
SYS$Time	system time (textual form)	4.6.2
SYS$Date	system date (textual form)	4.6.2
SYS$Version	Panos version number	
PROGRAM$Verbosity	controls feedback	4.9.1
PROGRAM$Help	controls help option	4.9.1
PROGRAM$Identify	controls identify option	4.9.1
PROGRAM$Force	controls permission override	4.9.1
PROGRAM$Confirm	controls confirmation requirement	4.9.1
PROGRAM$Abandon	controls action following an error	5.4
Alias$cmd	alias of cmd	2.7.2
Edit$...	for use by Editor	7.3.3
Link$...	for use by Linker	8.3.2

The PROGRAM$ variables control the global settings of certain options used with most Panos utilities.

2.7.2 Aliasing Commands

Alias$xxx is a special form for global variables that enables aliases or abbreviations to be set up for individual command lines to suit personal (or system) taste. Command files provide a more generalised, but more expensive mechanism.

If a global variable alias$xxx has value "yyy", where yyy is any string, then the command:

```
-> xxx args
```

will be interpreted as if the user had typed:

```
-> yyy args
```

For example:

```
.set alias$ed "edit -buffer 350000"
.set alias$cp Pascal
.set alias$print "copy -to printer: -from "
```

Note that the command line is re-evaluated after the substitution of one alias so that it is possible to alias an alias. It is also possible to construct an alias which allows parameter substitution; in this case, an alias string is made of two parts separated by the character `-'. The substring before the `-' is a keystring, the result of which will be used for substitution of the substring after the `-'. For example:

```
->  set alias$saydone     ".key S/l-echo<S> done!"
-> saydone file1
```

will cause the command `echo f i le1 done!` to be executed. See sections 4.8 for details of parameter substitution and the keystring.

2.8 Start-Up and Configuration Data

In addition to the values of global (environment) variables (see 2.7), the behaviour of the system is also governed by configuration data. These data relate to lower level implementation features such as keyboard auto-repeat rates, printer assignment, etc.), but unlike global variables they may not be set once Patios is running.

On initialisation, Patios reads two files: the configuration data file, `!Config' and the command file `!Panos'. Details of this process, and the default or initial values are given in Chapter 3.

3 System Organisation

3.1 Introduction

Context

This chapter describes the organisation of a Panos system: its constituent parts, how to configure it for a particular purpose, and how to start it up.

Details of installation and other introductory material may be found in the User Guide supplied with the system. Particular details referring to the organisation of certain utilities are given later in the relevant Chapters.

Organisation of this Chapter

First, details of configuring, loading and executing Panos are given. This is followed by a summary of system wide standard conventions. Finally the individual components of a Panos system are listed for reference.

3.2 Installation

The installation of Panos and its utilities is described in the User Guide supplied with the system.

3.3 Start-Up

Panos is loaded and run (`booted') from a filing system on which it has previously been installed.

On initialisation, Panos reads two files: the system configuration file, `!Config' which sets up a new configuration (this concerns data that relate to lower level implementation features such as keyboard auto-repeat rates, printer assignment, etc.); then it obeys the command file `!Panos'. Both of these files can be altered (although in different ways) from within Panos by the user to suit a particular system.

3.3.1 Loading Panos

Patios is loaded as follows:

1) If necessary, select a filing system where Patios may be found. This is only required if the currently selected filing system is different from that where Patios is stored.
2) Logon to the network if Patios is to be booted from NFS.
3) Run the Panos boot program.
4) Set the date and time using the Set utility.
5) Set up a local environment, e.g. for a particular package, if so required.

For example, to boot Patios from Winchester, type

```
*Panos
```

To boot Panos from the network, type for example:

```
*NET
*I AM kvk roff
*PANOS
```

Further details, including instructions for the DFS are given in the User Guide supplied with the system.

3.3.2 Pandora

Patios is booted from the Pandora command prompt. Patios itself, as currently implemented, relies upon Pandora which is a firmware kernel. It performs several tasks, including communicating with the I/O processor, (which will be some form of BBC Microcomputer). It issues a * prompt, and passes input from the user to the I/O Processor command line interpreter. Users have access to the usual * commands such as filing system commands, *FX, *TV commands and so on, as described in the *BBC Microcomputer User Guide* and elsewhere.

At present, the following programs run directly under Pandora:

Bas32 BBC BASIC,
Asm32 Code produced by the Assembler (as an option),
Panos Operating system.

Documentation of Pandora is of very specialised interest, typically to operating system developers only, and is found in the *Panos Technical Reference Manual.*

3.3.3 Configuration Data (!Config)

These are introduced in section 2.8. A full list is shown in Table 3-l.

The settings may be altered by executing the Configure utility, but their effect does not occur until Panos is re-initialised.

There are certain default values, shown in the Table, plus a set of values supplied as the initial values of the data file. These are not necessarily the same. The Configure program may need to be run when installing Panos onto a new system.

The file !Config is updated by the Configure utility at the directory root (`$') level. Separate customised !Config files can be created for a particular application on any hierarchical filing system, or for individual users on the Econet. When Panos is entered, it searches firstly in the current working directory, and then in the root ($) directory for !Config. Econet users can therefore have their own separate version of !Config which is stored in the `&' (logging-on) directory.

Full Details about many of the characteristics can be found in *BBC Microcomputer User Guide* or equivalent.

Table 3-1 Configuration Data

Item	Default (for ADFS)
Screen Mode	3
Screen Vertical Shift	0
Interlace	off
Keyboard auto-repeat rate	10
Keyboard auto-repeat delay	50
Caps Lock	off
Default RS423 format	8 data bits 1 stop bit No Parity
RS423 Receive baud rate	9600
RS423 Transmit baud rate	9600
Printer assignment	parallel
Printer ignore character	0
Max no of modules	1024
Physical filename of Panos database	$.PanosLib.PanData
Floppy disc drive speed	Fast
Size of global variable table	1500

The `Physical filename of Panos database' should point to `$.Panoslib.PanosDatafor an ADFS or NFS implementation; and , O.Pandata' for the DFS. Note that three different versions of !Config are supplied on the distribution discs: !Config (for the DFS), !ConfiA (for the ADFS/NFS), and !ConfiS for slow floppy discs (DFS). The file !ConfiA is automatically copied across from the distribution disc as $.!Config during installation onto the ADFS or NFS. Users of slow floppy discs (i.e. with slow access times) should replace !Config with !ConfiS before installation. (use *RENAME).

The `Size of global variable table' refers to the number of bytes allocated to the table which holds all of the Panos global variables. This would need updating if too many global variables were set.

The `Max no of modules' refers to the number of slots in the physical module table which are free for allocation for user program modules. See the *Panos Programmer's Reference Manual* for more details.

3.3.4 Start-Up Command File (!Panos)

This is a regular command file, which is executed when Panos is initialised. The version supplied with the system is used to set the values of global (environment) variables, (q.v.).

Command files, and the individual commands shown in the examples are described in 4.8. To alter !Panos, the file can be edited just like any other text file, (see Chapter 7).

The global variables initialised in the supplied versions are summarised in Table 3-2.

Table 3-2 Initialised Global Variables

CLI$Path	set up search path
EDIT$...	to configure the editor
LINK$...	to configure the linker
LL$...	to configure Pascal and C
PAS$...	to configure Pascal
C$...	to configure C
FILE$...	to set up file name extension mappings for ADFS and NFS
FILE$DFS...	to set up file name extension mappings for DFS
ALIAS$...	to set up command abbreviations

For an example, list or edit the version supplied on the system.

3.4 Standard Conventions

Many of the ways Panos may be organised and installed are `soft', that is may be implemented by the user. However, certain standard conventions are adopted in the software as supplied and in the documentation. Some of these conventions are described elsewhere in this manual, viz:

File Name Extensions	2.5.2
Standard Arguments	4.9

The conventions used for organisation of the discs into surfaces (for DFS) or directories (ADFS or NFS) in described in Chapter 3 of the User Guide supplied with the system.

3.5 Components

A Panos system has the following individual components:

- library of procedures organised into modules, memory resident (PanData)
- command line interpreter, memory resident (PanData)
- system loader (Panos)
- library of procedure names for resolving references (panos-lib)
- set of utilities including an editor and linker
- data files for initialisation (!Panos, !Config)
- system defined global variables (in **!Panos)**
- **set of install command files (Install-cmd, InstDFS-cmd)**
- **set** of language systems
- set of conventions (see 3.4)

4 Command Language

4.1 Introduction

Context

When Panos is entered, a system program (the command line interpreter) is given control. The Panos command line interpreter interacts with the user and, as its name suggests, has the function of accepting commands and executing programs, or interpreting command files. On system startup the interpreter is set running, it outputs a prompt, and waits for a command to be entered.

The Panos command line interpreter conceptually is an ordinary user-level program interfacing with Panos via the supplied library procedures (as documented in the *Panos Programmer's Reference Manual* and executing programs according to the procedural model. It is the central component in the user interface model.

Many of these user level aspects have an equivalent form for the programmer, and as such are described in the *Panos Programmer's Reference Manual* as follows:

Argument Decoding	Module DecodeArgs	Chapter 3
Data Formats and Conversions	Module Convert	Chapter 4
Time and Date formats	Module TimeAndDate	Chapter 10
Command Interpreter	Module Command	Chapter 15
Wild Symbols	Module Wild	Chapter 16

Organisation of this Chapter

The command intepreter has associated software for decoding arguments, (see 4.3), wild symbol expansion, (see 4.5), Further, there are associated conventions adopted where relevant by the utilities and compilers for standard argument strings, (see 4.9). These combine to provide a uniform model of interaction for the user.

4.2 Command Line Interpreter

4.2.1 Format of Commands

In interactive mode (as opposed to command file mode) user input to the command interpreter is in the form of text lines which have the format:

```
-> <CommandName>  <argument string>
```

where < CommandName > is one of:

(l) the name of a built-in'.' command
(2) the name of a file containing commands
(3) the name of a file containing a program

The program may be a user program, an application, or a system program such as a utility or a compiler.

The < argument string > is described in section 4.3.

The line may be edited as it is typed, e.g. to correct mistakes, by the use of special keys, see 4.4.

Commands and arguments are case insensitive as regards the user, (but not for the programmer - see 4.8.4).

4.2.2 Search Path

Except in the case of built-in commands the interpreter must first find the program or command file to execute. Since such files may in general be user or system supplied, and may reside in several different places depending on context (e.g. in several directories or disc drives), and will probably be in a different place from any data files, it is necessary to provide a special scheme for locating them.

The mechanism is that the interpreter searches a sequence of directories (or drives) until it finds a file whose name matches the command, (< CommandName >). The sequence or `Search Path' is set up by the user

by assigning a comma separated list of directories to the global variable CLI$Path, and then executing the NewCommand built-in command. For example:

(l) for DFS

```
-> Set CLI$Path dfs::0, dfs::2, dfs::1, dfs::3
-> NewCommand
```

(2) for ADFS

```
-> Set CLI$Path adfs:$.PanosLib, a
-> NewCommand
```

(3) for use with an application on ADFS

```
-> Set CLI$Path adfs:$.PanosLib,@.CadLib,
-> NewCommand
```

In example (l), the four floppy disc drives are searched in order, 0, 2, l, 3. In example (2), first PanosLib is searched, then the current working directory, although this could be the other way around. The latter case would provide user pre-emption of system programs at the probable cost of increased search time. In example (3), an application library has been added to the path.

The Set built-in command has been used in the examples, although the Set command with `var' parameter would be equivalent.

Note that the search path mechanism applies to commands only. Thus if the search path had been defined as in Example (1) above and the following were executed:

```
-> Set Dir dfs::1
-> Copy Fred -to vdu:
```

Copy would be searched for, but Fred must be in the current working directory.

Default search paths are set up by the supplied !Panos command file.

4.2.3 Command Prompt

The user is prompted by the Panos prompt ->. This symbol, which is the value of the global variable CLI$prompt may altered by use of the .set command or set utility. The string is evaluated before printing, (see 4.8.3) so, for example, setting it to the global variable sys$time will cause the current time to be used as a prompt. To include leading or trailing spaces, enclose the prompt in double quotes. For an example see section 2.7.

4.2.4 Action of the Command Interpreter

The action of the command interpreter can be summarised as follows:

1) The user is prompted via the control stream, (see 4.2.3);
2) A command is read from the control stream, (see 4.2.l);
3) The < CommandName > is extracted, and if applicable, the alias operation is carried out (see section 2.7.2). Non built-in commands are searched for (see 4.2.2);
4) Parameter substitution is carried out on the < argument string >, (see 4.8.3);
5) The remaining action depends on the type of command:
   Built-In commands are executed directly (see 4.7),
   Command files are obeyed by the interpreter (see 4.8),
   using the substituted argument string
   Program files are run (see below) - all Panos utilities will interpret the substituted argument string in a standard way, (see 4.9).

A program file is one in RIF format, i.e. contains relocatable machine code as produced by the Panos linker, such a file is loaded and executed. Program files should have the extension `-rif appended to the file name.

4.3 Arguments

The way in which the arguments following a command name are decoded and interpreted depends on the program being run. However all Panos

utilities use standard conventions and software to promote consistency and hence ease of use. All other programs are free to use the same scheme - which forms the subject matter of this section.

Note that argument decoding is `defined' by the `programmer', then `used' by the `(end-)user'. It is the latter who is of more concern in this section, the former in 4.8.4.

4.3.1 Position and Keywords

Arguments may either be positional or attached to a keyword. It is always legal to supply arguments attached to a keyword, but the use of position may be restricted in any given command, usually to `common', `obvious' arguments. Keywords, with the exception of state keywords, bind to the argument immediately following.

For example, the following uses of the Copy command all mean the same thing, i.e. copy file l to filet (thus overwriting it). In the examples, file l and filet are arguments, -from and -to are keywords.

```
-> copy -from file1 -to filet
-> copy -to filet -from file1
-> copy file1    -to filet
-> copy -from file1     filet
-> copy file1    filet
```

However, these are not all equally easy to read. The following example does not mean the same, but rather the other way around:

```
-> copy filet file1
```

The following are invalid:

```
-> copy -to filet file1
-> copy filet -from file1
```

Thus positional arguments must be given in the correct order; all arguments may be attached to a keyword in any order. The advantage of using keywords is that it is not necessary to remember the order, but the disadvantage is that more has to be typed.

Most commands have a small number of compulsory arguments followed by a large number of optional arguments. It is common practice therefore, both in definition and use, to specify positional notation for the compulsory argument(s) and keywords for the options, for example:

```
-> Pascal TEX -list printer:
```

Keywords can generally be abbreviated; the minimum abbreviation is specified on definition, although in Panos certain conventions are employed.

Note that the "-" identifying a keyword should not be confused with that separating a base file name from its extension.

4.3.2 Format of the Argument String

The < argument string > introduced in 4.2.l has a format as follows. For a full, definitive specification, see the *Panos Programmer's Reference Manual.* See also section 4.8.4.

Argument String

This is the string supplied by the user after the command name. It is a list of argument groups separated by spaces.

Argument Group

An argument string comprises one or more argument groups. An argument group is a list of one or more arguments separated by commas and associated with a keyword. In this publication, the term will be used to refer to the argument group plus the keyword (if present).

Argument groups may be optional or compulsory. The former are sometimes called options. Default values for arguments may be defined, and are adopted if not specified by the user.

Thus an Argument group is one of:

- a list of one or more arguments separated by commas
- a keyword plus a list of arguments
- a state keyword (see below)

Argument

This is a single item with one of the argument types listed in Table 4-l.

Table 4-1 Argument Types

Type	Examples
string	"->", "6-Oct-1985"
file name (includes devices)	TEX-pas, printer:
integer	12-10,42
cardinal	10
Boolean	true, false
name of STATE keyword	confirm, noconfirm

A state keyword has no arguments, but has two values, essentially true if the keyword itself is used, false if it is prefixed with NO.

Two standard state keywords are -help and -identify (see 4.9.l).

In this Guide the term argument is sometimes used loosely to refer to an argument group, providing no confusion will arise.

Keywords

A keyword is a single word that identifies a particular argument group. Alternatively, in some cases, an argument group may be identified through its position, (see 4.3.l). If a keyword is given, the character `-' must precede it (keyword `stropping'). The argument follows the keyword immediately, except in the case of state keywords that are in effect their own argument.

Keywords may be abbreviated, (see 4.8.4), and are case insensitive.

4.3.3 Examples of Command Lines

```
-> f77 -source prog -identify -opt +6
-> copy fl,f2,f3 -to vdu:
-> copy -files f1,f2,f3          -to file6 -force
```

In the first example:

f77	command name
-source prog -identify -opt + 6	argument string
-source prog	argument group (associated with -source)
-source	keyword
prog	argument (file name)
-identify	state keyword
-opt + 6	argument group
-opt	keyword
+ 6	argument

In the second example:

copy	command name
-from f1,f2,f3 -to vdu:	argument string
-from f1,f2,f3	argument group (associated with -from)
-from	keyword
fl	argument (file name)
f2	argument (file name)
f3	argument (file name)
-to	keyword
vdu:	argument (file name)

4.4 Line Editing

During input from the terminal (on the control stream), some keys have special meanings that enable them to edit the line being typed. Once (RETURN) is pressed the command is executed. These special meanings generally hold also during the execution of systems or user programs, but not necessarily so, for example in the editor.

A summary of these keys and their actions is given in Table 4-2. A **fuller description may be found, for example, in the** *BBC Microcomputer User Guide.* **In addition certain keys have a special meaning within Panos; these are listed in Table 4-3.**

Table 4-2 Editing Keys

DELETE	delete last character
COPY	copy character under read cursor
↑ ↓ ← →	move read cursor
CTRL - U	delete current line
RETURN	newline

Table4-3 Special Keys

CTRL - D	end of file
ESCAPE	generate an asynchronous event
TAB	move to next tab position
Function Key	bound to user defined value

Note in particular the effect of *ESCAPE* . In **general this will interrupt the current program and return to the level above, for example to the command interpreter. Thus this key can be used to `escape' from `incorrect' situations.**

Tab positions and function key bindings may be defined by the user by means of the Set command.

Keys auto-repeat at a rate and delay determined by the configuration file ! Config.

4.5 Wild Symbols

Many of the commands take one or more file names as arguments forming part of an argument group, for example as in:

```
-> copy -from file1,file2,file3                -to BigFile
```

In many cases the typing will be time-consuming or unreliable. Many Patios system programs, where appropriate, permit the use of `wild symbols' to act as abbreviations for file names. If there were only those three files starting with `file' in the current directory, then the above example could be abbreviated to:

```
-> copy -from file* -to BigFile
```

Further, long and descriptive, but hard to type, single file names can be abbreviated in this way. For example, providing there is no ambiguity, the following are equivalent:

```
-> set dir Releases.Notice10
-> set dir Re*.N*10
```

In general, the system will expand a wild symbol (or `wild card') into a single file name or a list of file names. The latter case will only be legal in certain contexts such as the first one above, but not in the second.

The full list of wild symbols is shown in Table 4-4. Strictly speaking, the first two apply to arbitrary strings, but in practice their use will be limited to file names.

Table 4-4 Wild Symbols

?	stands for any one character in a name.
*	stands for any string of zero or more characters.
...	means any arbitrary pathname.

See also 2.5.5 for special symbols used in file names.

Examples are:

test-rif	File name with no wildcard.
$.test-???	Three single characters (the file extension) are unspecified.
&...	All objects (files or directories) which are children of this directory (not the `&' directory itself).
$.Library.*	Any object in the $.Library directory.

Examples of use are:

```
-> set dir U*c
-> copy file?-* -to dfs:
-> cat adfs:$.*Lib,*-f77
```

4.6 Data Formats

Within the user interface of Panos, there are a number of objects that need to be represented. In particular users will be required in input data in a given format. In this section is a description of such formats for all objects, or a reference to the definition elsewhere in the Guide.

4.6.1 Simple Items

Primarily simple items are the values of arguments, i.e. strings, integers, cardinals, Booleans, and State keywords, see 4.3.2. This also includes file names, see 2.5.l.

4.6.2 Time and Date Format

The time and/or date is used by a number of utilities.

The time format is:

< hours > : **<minutes >** : < seconds > : **< centiseconds >**

the seconds and centiseconds being optional. Either 24 hour clock or 12 hour (+ am/pm) may be used.

Examples

10:13:07:67
: **means thirteen minutes, seven and sixty-seven hundredths of a second past ten.**

10:13 pm
: **means 13 minutes past lOpm (or 22:13).**

The Date format is reasonably flexible: `standard' and `textual' formats (and permutations) plus some extensions are permitted, (see the *Panos Programmer's Reference Manual* for a full definition.

For example:

9 Nov 85 1985-11-09 9th November 1985

all mean the 9th November 1985.
The form DD/MM/YY is not permitted as it is ambiguous (day and month may be interchangeable).

4.6.3 VDU **Characteristics**

Modes

Screen Modes are represented as the cardinals 0 to 135 inclusive.

Colours

Colours, both background and foreground are represented as one of the strings listed in Table 4-5.

Table 4-5 Colours

black
white
red
blue
green
yellow
cyan
magenta

or one of the above prefixed by `flashing-'.

Paged Mode

Paged Mode is represented by the Boolean true for paging, false for scrolling.

4.7 Built-in Commands

Command lines beginning with a `.' (after leading spaces have been removed) introduce commands which are built into the Command Interpreter. These are `primitive' commands in that they carry out low level or specialised tasks.They do not include provision for wild symbol expansion or use of search paths. In practice they are for the programmer (e.g. in command files); they need never be employed by the end-user.

The commands are shown in Table 4-6.

Table 4-6 Built-In Commands

. < SPACE >
: If the character after the . is a space, the rest of the line is ignored. This is useful for commenting command files.

.Delete < variable-name >
: Removes the global string from the environment.

. Help
: Provide help information on the built-in commands.

. key
: See command files, section 4.8.4.

.NewCommand
: Causes the Command Interpreter to use the value of CLI$Path to determine the future set of known commands. This must therefore be quoted after altering the CLI$Path, see 4.2.2.

.Obey < command file name > < arguments >
: Execute the named command file with the given arguments. This ignores the search path, i.e. a full path name must be given. This could be used, for example, to execute $.!Panos.

. pwd
Prints the current working directory.

. Quit
Leave Panos.

.Run < file name > < arguments >
Run the relocatable image in the named file passing it the given arguments. This ignores the search paths, i.e. a full path name must be given.

.Set < variable-name > < value >

Set the global string variable to the given value. A null value "" will cause the variable to be deleted from the environment.

.swd < path >
Sets the current working directory, see section 2.5.5.

. wait
This command causes the command interpreter to wait until RETURN is pressed before continuing. The main application is within command files which interact with the user; the `install' command files which install Panos onto the DFS employ this mechanism.

Examples

```
-> . comment
->  obey adfs:$.!Panos
->   run nfs:$.stamp-rif
->  swd dfs::1
->  set cli$stop -1
```

.set cli$path, newcommand and swd are often used together, for example after these commands,

```
->  set cli$path dfs::0
->  newcommand
->  swd  dfs::1.
```

the command:

```
-> cat data-dat
```

is equivalent to typing:

```
-> dfs::0.cat-rif dfs::1.data-dat
```

4.8 Command Files

Command files permit the user to store commonly occurring sequences of commands in a file and execute that sequence by issuing a single command. In this way the actual command sequence can be hidden to promote ease or convenience of use. Panos command files support command sequences, parameter passing and procedural calls (i.e. they may be nested).

The `use' of a command file is no different from using any other command; indeed this is a primary objective. This chapter is concerned more with the `definition' or writing of such files.Writing command files is a practical proposition for end-users as well as programmers, although there are complexities to be overcome in the more sophisticated examples.

Organisation of this Section

In sub-section 4.8.l the fundamentals of writing command files are presented along with some simple examples. Writing more complex command sequences involving the definition of arguments is more difficult and forms the subject matter of the remainder of this section.

4.8.1 Basic Facilities

If the name of a command typed in response to the Panos prompt is not a built-in command, it is assumed to reside in a file somewhere on the filing system.

If the file contains commands (indicated by the first character being a $ character) the commands are read and executed, and the file is said to be a command file. Command files should have the extension `-cmd' appended to the base file name.

Commands contained in a command file may be built-in commands, executable programs, or calls to further command files. Each line in a command file must start with $, and may be followed by any number (including zero) of spaces.

The use of help has a different interpretation in command files from that documented in 4.7, (see 4.8.4).

If the global variable cli$echo is set to `true' the command lines will be ‚echoed' on the screen as they are obeyed.

Termination Conditions

A command file is obeyed until some termination condition occurs. The first and simplest case is that the command sequence is completed.

Secondly, a user may terminate the execution by pressing the ESCAP key. In the case of nested command files, the use of ESCAPE during execution of a command terminates that command and returns control to the level above.

Thirdly, under certain conditions, an `error' during the execution of a program forming a command sequence will terminate that sequence. The mechanism is that each program returns a result code which is assigned to the global variable CLI$ResultCode. If the value of this is more negative than the value of the global variable CLI$Stop then the command sequence will be terminated. If not, then the next command in sequence will be obeyed. The default value of CLI$Stop is -64, i.e. stop on errors, but not on warnings.

See also Chapter 5.

Simple Examples

The examples demonstrate simple uses of command files, the first to set up a personal environment, the second to configure the system for an application.

```
$  Identify !Mark 21st August 1985
$  Set Alias$ls    "Catalogue -full -header"
$  Set Alias$ed    "edit -buffer 200000"
$  Set Alias$ty    "copy -to vdu: -from"
$  Set Alias$pr    "copy -to printer: -from"
$  set Alias$li    "lisp -image $.PanosLib.Lispimage -identify"
$  Set Program$Verbosity 99
$ Show var sys$*
```

```
$  Identify ADFS !GCaL Version 1.00/01
$  Help Initialises GCAL on ADFS
$  set gcal$Lib                adfs:$.GCaLLib
$  Set Cli$Path                adfs:$.PanosLib, adfs:&.$.GCaLLib, @
$  NewCommand
$  set File$-gcal              _gcal.-
$  set File$-gout              -gout.-
$  set File$-glib              _glib.-
$  set File$-gfl               _gfl.-
$  set File$dfs:-gcal          u.-
$  set File$dfs:-gout          V.-
$  set File$dfs:-glib          g.-
$  set File$dfs:-gfl           u.-
```

4.8.2 Parameters with Command Files

The above simple examples illustrate the use of command files without arguments. To be more useful it is necessary to have mechanisms by which:

(a) the user can supply an argument string with the name of a command file
(b) a legal format for that string can be defined by the author of the command file
(c) the system can decode the argument string and check it for legality, (with the same rules as in programs)
(d) the values of arguments can be passed onto the individual commands that make up the command file and used as arguments to them

In this section an illustration of the above mechanisms is given. The example is a simplified version of the f77 (FORTRAN 77 compiler) command which hides from the user the need to call two separate programs in sequence. These are the front-end (f77fe) and the code generator (f77cg).

```
$  Identify Fortran Command file 1.10/01
$  key source/e-f77 list/s opt/k[+]
$  Help
$  Help -Source          Source file
$  Help -List            Enable listing
$  Help -Opt             Compilation options
$  Help
```

```
$  set CLI$Stop -64
$ f77fe <source> <List> -opt <opt>
$ f77cg <source> -opt <opt>
```

The following are sample uses of the above command file definition:

```
-> f77 Spice
-> f77 Spice -list
-> f77 -source Spice -List -opt +tW0
```

This works as follows:

The user types a command with its associated argument string in the usual way (a).

The lines `.help' and `.identify' simply provide the user with information.

The line `.key...' is an example of a keystring. This defines the argument string format (b). It specifies that there should be three argument groups, source, list and opt. The first is a file name (/e), the second is a state keyword (/s), and the third is a string, but must be supplied with a keyword (/k).

The command interpreter checks the actual argument string against this definition (c).

The actual arguments supplied by the user are substituted for the place markers shown in angle brackets, (see (d)). For example the argument associated with the keyword -source is substituted for < source > .

Therefore the three examples of use translate into:

```
f77 Spice
    -> f77fe Spice-f77 -opt +
    -> f77cg Spice-f77 -opt +

f77 Spice -List
    -> f77fe Spice-f77 -List -opt +
    -> f77cg Spice-f77 -opt +

f77 -source Spice -List -opt +tWO
    -> f77fe Spice-f77 -List -opt +tW0
    -> f77cg Spice-f77 -opt +tW0
```

4.8.3 Parameter Substitution

As outlined in 4.2.4, the Command Interpreter performs parameter substitution on all lines. This is carried out on all argument strings but before the arguments are decoded by the program or command file.

Its main significance is in command files, although it is of wider applicability. See the *Panos Programmer's Reference Manual* Chapter 3 for full details.

Parameter substitution enables actual argument values supplied on the command line by the user to be substituted for formal values within a command file. In addition global values (from global variables) may be substituted.

The following rules are applied:

- a parameter is a word enclosed in < > brackets. Leading and trailing spaces are stripped.
- if in a command file then the parameter word is first looked for in the decoded arguments derived from the'. Key String', (see 4.8.4). If there is a first line beginning with `.key', then a string representation of the argument is substituted for the parameter in the line.
- the parameter word is looked for in the global environment strings. If found its value is substituted.
- if no substitution has occurred then an error is generated.

4.8.4 Argument Decoding and the Keystring

It is possible to include argument decoding in command files, similar to that provided by the Panos run-time library. An abbreviated account is given in this section; for full details, refer to the *Panos Programmer's Reference Manual).* To make use of this facility.key must be included as the FIRST line of the file (comment lines are not excluded from this rule). For example, see 4.8.l.

The keystring describes the arguments which the program expects. This process is illustrated in figure 1.

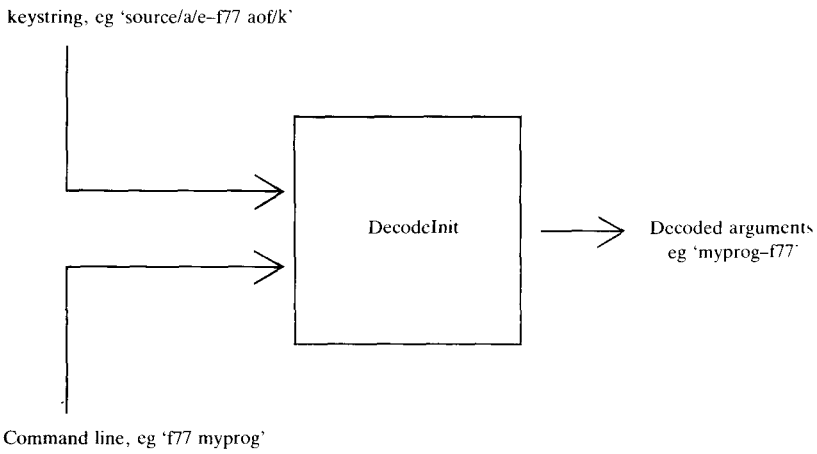

Figure 1 Argument Decoding

The Keystring

A keystring is a sequence of keywords, (separated by spaces or commas) which are qualified by control characters called option specifiers (e.g. /k and /e in the example of 4.8.l). These determine the type of keyword, and the number and type of arguments that may be associated with it.

Also associated with each keyword is an optional default argument list. This is used if the user does not supply any arguments on the command line for that keyword. The case of the keyword determines the minimum abbreviation. Upper case specifies compulsory characters. Thus NAme is matched by NAME, NAM, NA, but not N.

Option Specifiers

There are three classes of option specifier:

l) Quantity option

This is used to indicate the number of arguments which may be associated with the keyword. There are three formats:

/ < number >	This specifies that at most < number > arguments may be associated with the keyword.
/ _ < number >	This specifies that exactly < number > arguments must be supplied for the keyword.
/?	This specifies that any number of arguments can be supplied.

If no quantity option is supplied then /l is assumed, i.e. keywords are expected to have one argument by default.

Some examples are:

Keystring	argument groups matching
INPUT/3	-input x,y,z -INPUT x -input
INPUT/=3	-input x,y,z
INPUT/?	-input w,x,y,z -Input
INPUT	-input x

2) Type option

This option indicates what type of arguments are expected to be associated with a given keyword. Possibilities are:

/I	Integer. This indicates that integer arguments will be used with the keyword.
/C	Cardinal. This indicates that cardinal (positive integer) arguments will be used with the keyword.
/B	Boolean. This indicates that two argument values are possible: TRUE or FALSE.
/E[-ext]	Extant. This means that the keyword's arguments are expected to be files residing on the filing system. A check is made that the names provided exist. In addition, an optional extension may be given which is automatically appended to file names which do not have an extension already.

/R	Rest of argument string. The value will be the string made up of all the characters to the end of the argument line with no further interpretation.
/L	Literal string; that is the string made up of all the characters up to the next explicit keyword with leading and trailing spaces removed.

If no type option is supplied then the keyword is interpreted as a plain string.

Some examples are:

Type	Keystring	argument groups matching
Integer	POSITION/I	-position 128 -position 16_1A -position -3024
Cardinal	LENGTH/C	-length 128 -length 16-1A
Extant file	-SOURCE/e-f77	-source prog (value is prog-f77) -source prog-f77 (value is prog-f77) prog (value is prog-f77)
	SOURCE/e	-source prog (value is prog) -source prog-f77 (value is prog-f77)

3) Keyword Presence

There are two options to control the `parsing' of the key string, compulsory argument, and compulsory keyword, plus three to control the detection of keyword names.

Compulsory Argument	/A	This implies the keyword must have at least one argument (although the keyword itself need not occur). The keyword cannot be used with a default argument list, (see below).
Compulsory Keyword	/K	This means that the keyword can only have arguments if the keyword itself is also given.
Presence	/P	True if the keyword is present, false otherwise.

Non-Presence	/N	Permits NO to be pre-fixed to /P options.
State	/S	This indicates a state keyword. These don't have arguments supplied by the user, but are interpreted as TRUE if cited in the command line and FALSE otherwise. Such arguments can take neither default arguments, quantity options, nor the /A option. Equivalent to /K/P/N/ = 0

Examples:

OBJECT/a/e-aof	-object x	
	-object x-aof	
	x	
	x-aof	
LIST/k	-list 1file	
	but NOT Ifile	
LIST/s	-list	value is TRUE
	-nolist or < blank >	value is FALSE

Default Values

A value in square brackets `[' and `]' in a keystring is interpreted as a default value, i.e. is used if the user does not provide any.

Example:

FILES/?[f1,f2,f3]	-files oneF,twoF	value is oneF,twoF
	-files	value is f1,f2,f3
	(blank)	value is f1, f2, f3

/S keywords have implicit default arguments.

/A keywords cannot have default arguments.

Assignment of arguments to keywords

This sub-section describes how the user-supplied arguments are interpreted using the programmer-supplied keystring. A more detailed description is given in the *Panos Programmer's Reference Manual.*

The rules for interpretation are:

a) Argument groups qualified by the keyword name can be supplied in any order.

b) Argument groups not qualified by the keyword name are assigned (essentially as values to keywords) from left to right.

   Note that, in this process, /K (compulsory keyword) and /S (state) keywords in the keystring are ignored.

c) The character'-' must be immediately followed by a keyword name.

d) If the user does not supply a value, and a default value is present in the keystring then that default value is assigned to the keystring.

e) The keyword names -help and -identify are special, in that the program will always respond to them, even if the rest of the command line does not make sense. This allows users to type, for example, f 77 -He l p and receive some help information. It is not necessary to specify these keywords explicitly in keystrings.

4.9 Standard Conventions

This section describes the conventions adopted by all supplied programs for argument strings. Users and software developers are free to adopt whatever conventions they wish but are encouraged to adopt those described here.

There are many arguments that the systems programs have in common. To avoid duplication in the individual descriptions, these standard arguments are described here.

Square brackets indicate that the keyword is optional (i.e. positional notation may be used). Upper case signifies minimum abbreviation.

Many of the options are state keywords, i.e. are true if the keyword is given, or false if the prefix NO is followed by the keyword, for example -Confirm is true, -NoConfirm is false. The action is carried out if the result is true.

If the state keyword is not given, then the default value is used. The default value is false unless otherwise stated, but see 4.9.3.

In 4.9.1 certain standardly interpreted (but not necessarily universal) arguments are listed. In 4.9.2 the usage of such arguments in certain classes of system program is described.

4.9.1 Standard Arguments

[-Help]
: Typing -Help on the command line will make the program print a summary of its arguments, and examples of their use. If this argument is given, the program itself is not executed.

[-IDentify]
: Issuing this option prints the full path-name of the program, followed by its version number. The program is executed.

[-ERRor] name
: `Errors' may be generated in a variety of ways, for example, a file name may be misspelt. The error messages (and any verbose information generated by the program) are sent to a Panos stream called `error:', (see 2.2). Usually, this stream is associated with the screen. However, errors can be redirected to other files or devices by following the keyword -error with a name, e.g. -error printer:

[-CONTrol] name
: This is similar to -error but is used to redirect input from the currently selected control stream to another stream, (see 2.2). Any program which issues prompts reads the reply from the control stream. Usually, this stream is associated with the keyboard (`kb:'). Prompts are written to the output stream associated with the control stream, or the screen if there is no associated stream.

[-Verbosity/Verbose]
: Feedback arising from actions performed by the program e.g. deletions, file creations, copying etc. is given on the error stream if -Verbosity is present or the value of global variable Program$Verbosity

is greater than 0. Only one-line error messages (usually from the system) are sent if the verbosity level is zero. The higher the level of verbosity, then the more unimportant operations are reported. The default value is 3.

[-Confirm]
If -Confirm is present or Program$Confirm is set to `True' then confirmation of the action about to be taken by the program (and any unforced deletion necessary to perform that action) will be required. The default setting for this argument is `noconfirm' except in the delete utility, but see 4.9.3.

[-Force]
If `-Force' is present or Program$Force is set to `True' then any locked files will be unlocked. The `-Force' keyword must be specified to overwrite existing files. Compare this with `-Confirm', which will overwrite existing files, but will request confirmation before doing so, but see 4.9.3.

[-Abandon]
If `-Abandon' is present or Program$Abandon is set `True' then the program will be exited if any error is detected, otherwise it will continue until a fatal error occurs, but see 4.9.3. This is of particular use in command files. See also 5.4.

[-INput]
If input to the program is read from the input stream, input: (see 2.2), then use of this option enables it to be redirected so that it is read from the specified device or file. By default input: is the keyboard.

[-FROM]
A synonym (alias) for -input

[-OUTput]
If output from the program is sent to the output stream, output: (see 2.2), then use of this option enables it to be redirected so that it appears on the specified device or file. By default output: is the screen.

[-TO]
A synonym (alias) for -output

4.9.2 Usage of Standard Arguments

All Programs

All programs use the following (see 4.9.l):

-help
-identify

They are interpreted in a special way by the argument decoder so it is not necessary to specify them in a keystring.

Utilities

In this context, utility program means those described in Chapter 6. Most of these utilities use the following (see 4.9.l):

-help
-identify
-error
-control
-verbosity
-confirm
-force
-abandon

Language Compilers

For details of the action of each compiler, see the relevant language reference manual. BBC Basic is an exception since it does not use Panos.

All compilers use the following (see 4.9.l):

-help
-identify
-error

In addition, most compilers use the following:

-source	for the source text
-aof	for the object module
-list	for the source listing

Compilers follow a set of rules for processing the file extensions:

1. If an extension is given in the keystring then it will be appended to the file name if the user supplies no extension.

2. The value of the argument is the file name with the extension added.

The compiler is expected to generate names for other files required from the source file name with its extension removed.

For example, using the f77 compiler:

The extension used for source files is `-f77'

```
-> f77 myprog
```

This command compiles the FORTRAN source in myprog-f77 and places the object output in myprog-aof.

```
-> f77 myprog-txt -list
```

Compile the FORTRAN source in myprog-txt place the object output in myprog-aof and the listing in myprog-lis.

```
-> f77 myprog -aof temp -list printer:
```

Compile the FORTRAN source in myprog-f77 place the object output in temp and send the listing to the printer.

4.9.3 Global Control

The utility programs all take a number of options as described in 4.9.l. Many options have default values. For example, overwriting a file will by default require confirmation. This may not be to the user's taste.

These default values may be changed by the user through the use of PROGRAM$... global variables, (see 2.7). This facility enables users, either temporarily or permanently, to customise the global behaviour of the system. In the example above, the default may be changed to no confirmation for all programs thus:

```
-> set var PROGRAM$Confirm      FALSE
```

This mechanism governs global behaviour. Local behaviour within a single invocation of a command can of course be defined by means of the option. For example, the use of feedback may be governed for all programs by setting the value of PROGRAM$Verbosity, or for a single program through the -Verbosity option when issuing the command. The latter always takes precedence over the former.

The PROGRAM$ variables are listed in Table 4-7.

Table 4-7 PROGRAM$ Variables

PROGRAM$ Verbosity	Cardinal
PROGRAM$Help	Boolean
PROGRAM$Identify	Boolean
PROGRAM$Force	Boolean
PROGRAM$Confirm	Boolean
PROGRAM$Abandon	Boolean

5 Feedback and Errors

5.1 Introduction

Context

Feedback is given to the user for a number of reasons, usually to provide information on the state of the actions being carried out. A particular form of feedback is notification of errors. Often, an error condition will terminate the current action.

Error handling forms the topic of Chapter 2 of the *Psion Programmer's Reference Manual.*

Organisation of this Chapter

In this chapter, the different ways in which errors are reported forms the topic of the first section. This is followed by a description of the ways in which feedback may be given. Then comes a summary of the ways in which the user may control the effects of errors and perhaps recover from them. Finally, there is a summary of the help facilities.

5.2 Error Messages

In this context, an error is taken to be a condition that ordinarily results in the termination of a system or user program. Generally these are the `fault' of the user of the program.

The system reports different classes of user error in a number of different ways.

In some cases the `error' is considered as no error at all, perhaps it is the null case. No error message is given. For example:

```
-> show var NoVar
```

In **other cases the running program will detect the error, for example, a filename may have been mistyped:**

```
-> copy NoName -to vdu:
Error in copy : failed to find 'NoName'
```

In further cases the error may be detected and reported by a library procedure called by the running program. In such a case a message is given preceded by three + signs. The Panos library module responsible identifies itself but this is **unlikely to be of interest to the typical user. For example:**

```
-> set var sys$time xxx
+++ Can't set 'sys$time' - reserved variable
+++ Detected by module GLobaLString

-> copy -NotKey NoName -to vdu:
+++ Keyword '<NotKey>' not known
+++ Detected by module Parameter
```

Compile time and run time error messages from user programs under development are described at length in the relevant language reference manuals.

A full list of Panos error messages is given in Appendix A of the *Panos Programmer's Reference Manual*

5.3 Feedback

Error messages are a particular form of feedback. Other forms of feedback include warnings, or simply the commentary given showing progress with or completion of an action. Although this commentary can be of value, especially to inexperienced users, it can be tedious. Therefore in Panos the user has some control over its `verbosity'.

The command option -Verbosity and the global variable Program$Verbosity (see 4.9.3) are conventionally used to govern whether the program provides a commentary through strictly unnecessary but reassuring feedback. The level of warning and error message reporting may also be set in this way.

5.4 Error Control

There are a number of ways in which users can control the way in which errors are processed. Mostly, these are documented elsewhere, but referenced here.

Abandon

-abandon and PROGRAM$Abandon (see 4.9.l) are used by the utilities to determine whether a single error condition should terminate the invocation of the command. For example, if a list of files were to be copied, should the remainder be copied if one does not exist. The result code (see below) will be set accordingly.

Command File Termination

The global variable CLI$Stop (see 4.8.l) is used to determine whether a command file should terminate following an `error' in one of the sequence of commands.

Verbosity

-verbosity (and PROGRAM$Verbosity) (see 4.9.l) control the level of warnings and other feedback given.

Program Results

The global variable CLI$ResultCode conventionally takes on one of the following values on completion of a program:

+ ve	available for users
0 to -63	warnings
-64 or less	errors

Command files are terminated if this value is more negative than the value of CLI$Stop.

Event Handling

Both asynchronous and synchronous events (e.g. errors) can be trapped by the user program and suitable action taken. For details see 2.6 or the *Cxenos Programmer's Reference Manual.*

ESCAPE

Pressing the ESCAPE key is an asynchronous event. Conventionally it returns control to the calling program.

5.5 Help Information

On-line help information is provided in a number of ways.

Help Command
: The Help command describes the usage of the program given as its argument. For example, to gain help on the use of the linker, type:

```
-> Help linker
```

Typing Help without an argument lists all the system programs (for which individual help exists).

Help Option
: Exactly the same message as above can alternatively be obtained by use of the -help option (see 4.9.l). For example:

```
-> Linker -help
```

Note that standards exist for the form of such help messages.

Built-In Help
: For help on built-in commands type help (see 4.7).

Editor Help
: The editor has a special help system, see 7.6.l.

6 Utilities

6.1 Introduction

Context

Panos relies upon utilities to perform most tasks. This approach has the advantage that Panos can be enhanced simply by writing new utilities to perform a particular function.

The utilities provided as standard with the system are described in this chapter with the exception of the editor and linker which are in Chapters 7 and 8 respectively.

There is also a small set of built-in commands which can perform some of the same tasks as the utilities; these have specialised uses, and are documented in section 4.7.

Conventions in this Chapter

A list of argument (group)s is given for each utility. All keywords are optional (and therefore enclosed in square brackets); however, the arguments which they refer to may not be optional, e.g. the logon utility does not require the keyword `-as' to be stated, but logging on details must be specified.

Each utility has its default settings for state keywords. For example, the delete utility has `-confirm' set by default, although in others it is `-noconfirm'.

The options listed in 4.9.l are available for almost all the utilities in this chapter, and are therefore not described further.

Demonstration

A selection of the utilities is shown in figure 2.

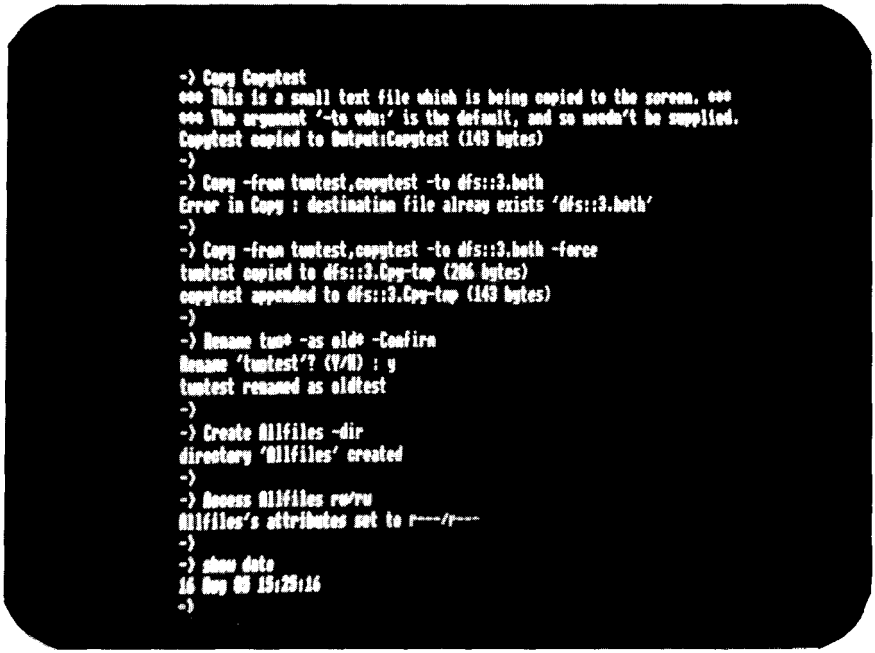

Figure 2 Utilities Demonstration

Organisation of this Chapter

Each utility is simply listed in turn, in alphabetical order.

There are no special sections on command language, feedback and so on since the standards described in earlier chapters apply. The utilities are installed and configured as part of Panos, see Chapter 3.

6.2 Access Command

This utility allows the user to change the access permissions of a list of one or more files.

Concepts

See 2.5.4.

General Form

-> access arguments

Arguments

[-FILEs] file names
: The list of files to be affected is optionally preceded by the -FILEs keyword. Wildcards may be used.

[-ATTRibutes] attributes
: The attributes (permissions) are optionally preceded by the -ATTRibutes keyword. They depend on the physical filing system and take the same format - (see the relevant filing system user guide) The utility will not enable the `E' attribute to be set on the ADFS and will not affect files which have this set.

[-BEFORE] date
: Only the files in the -FILEs list with no datestamps or datestamps before the given date will be copied. Dates can be given in most unambiguous formats, and optionally include a time, e.g. Thursday 20th June 84 8:30am. See 4.6.2 for a definition of the format. Note that `Today' is a valid date. If no time is specified then the start of the day is taken.

[-AFTER] date
: See -BEFORE (substituting after for before).

The standard arguments for utilities are also available, see 4.9. t-2.

Examples

```
-> access Dave-pas rw/r
-> access * rw/r
-> access -files fred,jim,sheila -attributes rl/ -after 10th-mar-81
-> access -file jam wr
```

6.3 Catalogue Command

This utility obtains information about files. The amount of information printed can be varied from a simple list of file names, to complete pathnames with the associated file information (access permissions, file creation date etc). This utility is shortened (aliased) to `cat' in the !Panos start-up files provided with the system.

The exact form of the information depends on the filing system.

See figure 3 for a demonstration.

Concepts

See 2.5.

General Form

```
-> Catalogue arguments
-> Cat arguments
```

The first form is for the ADFS and NFS only.

Arguments

[-FILEs] file names
: A list of zero or more files or directories to catalogue. The normal file name conventions apply (i.e. wildcards may be used etc.)

[-Depth] n
: Controls the depth of nesting of the listing. The keyword is followed by a number n which gives the maximum level to which directory contents are expanded in the filing system hierarchy. A value of zero gives no expansion. The default is one.

[-FULL]
: Gives file attributes in addition to the name. This information includes access permissions, date of creation, and whether the file is a directory.

[-Header]
Causes the pathname of the directory to be printed just before the list of files it contains.

[-COLumns] n
Forces the output into n columns. By default the number of columns is chosen according to the longest name to be output. This option has no effect if -full has been specified.

[-BYName]
Sorts the output into strict alphabetical (ASCII) order.

[-BYDate]
Sorts the output into chronological datestamp order. Unstamped files are considered to be the most recent, and are sorted alphabetically.

[-Reverse]
This reverses the order of output, after any sorting option has been considered.

[-TO] name
See 4.9.l.

The standard arguments for utilities are also available, see 4.9.l-2.

Examples

```
-> cat
-> cat lib
-> cat
-> cat *-f77 -to printer:
-> catalogue $...
-> cat *-f77, *-aof
-> cat lib??? -bydate -depth 2 -full
```

The 1 st example lists the current directory.
The 2nd example lists a directory called lib (in the current directory).
The 3rd example lists all objects (files and directories).
The 4th example lists all Fortran files on the printer.

Functions

By default, output is ordered in the natural order of expansion. That is, directory contents are listed alphabetically, and any listing of subdirectory contents occurs recursively at the end of the parent directory.

The -FILEs argument is an optional list of files to be listed. As with all 'file-type' argument groups, the names are separated by commas, and can include wildcards. If a file name is a directory, its contents are listed, rather than the name of the directory (but see -Depth).

The 1st example differs from the 3rd in that any sub-directories listed by the 3rd will have their contents listed, not their names. For example, suppose that, on the ADFS, the current directory (which is, for instance, the start-up directory `&') contains two files, linkfile and libdir, of which libdir is a directory containing the two files Tred' and `petunia'. Catalogue on its own will list:

```
libdir linkfile
```

The command `cat *'will list:

```
linkfile
Directory: 'ADFSA.libdir.fred' Date: 07 May 87 10:00:07
Directory: 'ADFSA.libdir.petunia' Date: 07 May 87 10:00:07
```

Note that the non-directory files in the list are given first, followed by the contents of any directories.

The option -Depth is used to prevent (or enable) directories' contents being listed.As mentioned above, if a given file name argument is a simple file, catalogue lists that name. If it is a directory, the contents of that directory are listed instead of its name. The -Depth option controls the depth to which directories are listed.

To prevent directories from being expanded into their contents, use -Depth 0. Using the structure above, typing

```
-> catalogue * -d 0
```

will prevent lib from being expanded and will display:

```
libdir linkfile
```

The numeric argument after -Depth specifies how deep into the hierarchy catalogue should look. The default is -Depth l, which causes it to list directories one level down from the argument directory. Greater values cause it to look further down the structure. Taken to the extreme, this can be used to list the whole structure of the disc, for example:

```
-> catalogue $ -Depth 99
```

will list files down to level 99 (in practice, ten is above the limit that people will use) starting from the root, $.

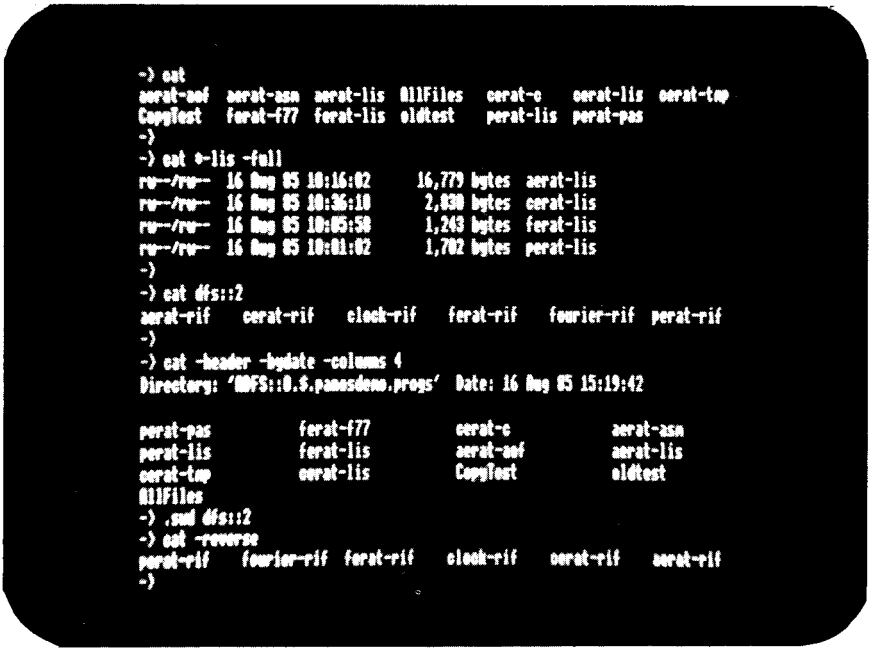

Figure 3 A Detuoustratiou of Catalogue

The -Header argument causes catalogue to print the names of directories before their contents. For example, typing

```
-> catalogue -header
```

with the example file structure would give:

```
Directory: 'adfs:&' Date: 02-feb-87 15:47:56
libdir linkfile
```

6.4 Configure Command

This utility allows the user to set up configuration options, i.e. alter the value of !Config.

Concepts

See 3.3.3.

General Form

```
-> Configure arguments
-> Config arguments
```

The latter form is for DFS, the former for ADFS or NFS.

Arguments

[- New]
: If this optional parameter is specified, a new !Config file is created in the current directory. An error message is therefore not given if there is no !Config file already in existence.

The standard arguments for utilities have no function.

Examples

```
-> configure
-> configure -new
```

Command Language

The utility consists of two modifiable screen pages, the second page containing attributes which should be altered with care.

A special command language is used with this utility. t and I are used to select a new item. Use - or i to select a new value for that item.

Help information about the modifiable characteristics is accessible on-line, and visible at the top of the screen. To move from page 1 to page 2, press SHIFT - F0, i.e. whilst holding SHIFT , press F0 (CU to get back to page 1).

Functions

If the !Config file is updated then the old file is renamed as !OldCon; this is as a safeguard in case the updated !Config file has been altered erroneously (for example, the auto-repeat rate may have been set too high), making it very difficult to re-create !Config.

When `Configure' is used, the file !Config is overwritten. The utility looks for an old version to update, searching (in this order) in the current directory, the start-up directory `&', and the root directory `$'. If no !Config is found, an error is given.

6.5 Copy Command

This utility takes a source list of one or more files, directories, and/or devices, and copies them to a destination, either a file or a directory.

Concepts

See 2.5.

General Form

```
-> Copy arguments
```

Arguments

[-FROM] name
: The -FROM keyword may optionally precede the list of objects to be copied. See 4.9.l.

[-TO] name
: This optional keyword is followed by the destination file name, directory name or device name. The default is to output: (usually vdu:). See 4.9.l.

[-Delete]
: If this switch is specified then the source file will be deleted after the copy has taken place (in effect renaming across filing systems).

[-CONTENTs]
: If the source list contains a directory then the directory contents will only be copied if -CONTENTs is specified.

[-Exact]
: Usually the file created by `copy' is datestamped with the current date. If this flag is specified, the datestamp information from the source file is used instead. When concatenating, the datestamp of the first file in the source list is used. This option is also used when copying non-Panos files, i.e. when the `date stamp' is actually a load/execution address. See also 2.5.3.

[-AFTER] date
: This argument takes a time and date string which when specified, means that only files with a later datestamp will be copied. This is useful for only backing up recently changed files. Note: this applies to files with datestamps, those without will be copied regardless. See 4.6.2 for the date format.

[-BEFORE] date
: See -AFTER (substituting before for after).

The standard arguments for utilities are also available, see 4.9.l-2.

Examples

```
-> Copy Filet -to Filet
-> Copy $.OldDir.File1 -to $.NewDir
-> Copy Filet
-> Copy dfs:File1 -to@
-> Copy dfs::O.File1 nfs:
-> copy -from kb: -to newfile
-> copy -from adfs:thisdir -to nfs:thatdir

-> copy *-cmd -verbosity 3
-> copy *-rif -to old*-rif -after 15th August 1984 7:30 am
-> Copy dfs::2.*-pas -to nfs:&.newfiles -after today
-> copy * to adfs:newdir -confirm
-> copy kb:first,kb:second -to adir -force
-> copy AnExecFile -to dfs::0 -exact
```

Functions

The action taken depends on the contents of the source and destination lists. If no destination is specified then the source is copied to the currently selected output stream which is usually (and by default) the screen, (3rd example).

The most typical use of this utility is moving files between filing systems, or from one location to another within a directory structure.

If the destination is not an existing directory, then the source files are concatenated into it, e.g. as in:

```
-> Copy -from File1,File2 -to &.Bothfiles
```

Beware when copying the entire contents of a directory into one other object: if this is not an existing directory, then a very large single file will be created. For example,

```
-> Copy Direct.* -to NotaDir
```

will concatenate all files in directory `Direct' to one file `NotaDir'.

The -CONTENTs keyword causes the directory's entire tree structure (if the source is a directory), to be copied to the destination directory or device retaining the same structure and names.

If both the source and destination are wildcarded, then matched wild-carded parts in the source list are substituted for the corresponding wild-carded parts in the destination. This facility can be used for backing up certain files, e.g.

```
-> copy *-rif -to old*-rif
```

6.6 Create Command

This utility allows the user to create directories (or files). It may be viewed as a `Copy' utility which doesn't take any source files. It creates a list of new files or directories, optionally of a given length. Its primary use is to create new directories on a hierarchical filing system.

Concepts

See 2.5.

General Form

```
-> Create arguments
```

Arguments

[-FILEs] file names
: This is the list of file(s) to be created.

[-Dir]
: If this argument is quoted the file created will be a directory.

[-Size] n
: This is the size in bytes with which a (non-directory) file should be created. The default is zero bytes, i.e. a null file.

The standard arguments for utilities are also available, see 4.9.1-2.

Examples

```
-> create neudir -dir
-> create some,many -verbose 1
-> create text -size 64
```

Functions

If -Force is set then any part of the path name which does not exist will be created.

6.7 Delete Command

This utility enables the user to delete one or more files or directories.

Concepts

See 2.5.

Geiieral Forin

-> Delete arguments

Arguments

[-FILEs] file names
This is a list of one or more file names. As usual, wildcards may be used in place of an explicit list. Directories may only be deleted if they are empty.

[-BEFORE] date
Delete the file only if its datestamp is before the date and time specified (or if it does not have a valid datestamp). See 4.6.2 for details of the date format.

[-AFTER] date
Delete the file only if its datestamp is after the date and time specified (or if it does not have a valid datestamp). See 4.6.2 for details of the date format.

The standard arguments for utilities are also available, see 4.9.l-2.

Examples

-> delete oldfite-f77
-> delete * -noconfirm
-> delete *-rif -force
-> delete * -before 15-Aug-85

```
-> delete oLdfiLe-*
```

Functions

Delete takes a list of file names to delete. Usually locked files will not be deleted, but this may be `forced' using the standard `-Force' option. Note that files specified on the command line are deleted in reverse order, so to delete an entire non-empty directory including both the directory and its contents, type

```
-> Delete dir,dir...
```

NOT

```
-> Delete dir...,dir .
```

By default, the standard keyword `-Confirm' is always set, i.e. confirmation is always expected before an object is deleted. To override this, use the `-NoConfirm' keyword.

6.8 Echo Command

This utility allows the user to perform parameter substitution on its arguments thereby evaluating them.

Concepts

See 4.8.3.

General Form

```
-> Echo arguments
```

Arguments

[-Lines] or [-Nl]
: Blank lines can be printed after the material has been echoed by either specifying -Nl (newline) or -lines for however many blank lines are required.

[-TO] name
: Usually `echo' sends its information to the standard output device, i.e. the screen. By specifying this option the user can cause the output to be sent to some other device, e.g. printer: or a file.

The standard arguments for utilities are also available, see 4.9.l-2.

Examples

```
-> echo a few words
a few words

-> echo cli$path's value is <cli$path> -nl
cli$path's value is dfs::0, dfs::2
```

Functions

This program takes a single argument (which may include spaces and punctuation such as commas) and prints it on the screen. The command interpreter carries out parameter substitution, and therefore global variables and arguments in a command file may be evaluated.

The following command file illustrates this process. It provides a simple version of the `catalogue' utility.

```
$   key files/e/?C*7
$ echo <files>
```

6.9 Logon Command

This utility allows the user to log on to a network file server, and is thus the Panos equivalent of the *I AM command.

Concepts

See *Econet File Server User Guide.*

General Form

```
-> logon arguments
```

Arguments

[-AS] logging on details
: This keyword (itself optional) is followed by the station number, user id and password (or subset thereof), constituting the body of the logging on command.

[-Pass] password
: If this optional argument is quoted, then the user will be prompted (on the next line) to enter an un-echoed password.

The standard arguments for utilities are also available, see 4.9.l-2.

Functions

This command sets the working directory on the network filing system (nfs:) to be the specified log-on directory (nfs:&). It does not otherwise change the current working directory.

Examples

```
-> logon lionel
-> logon -as fred secret
-> logon 4.126 panosthings -pass
```

6.10 Rename Command

This utility allows the user to rename a list of files or directories.

Concepts

See 2.5.

General Form

```
-> Rename arguments
```

Arguments

[-FROM] file names
: This keyword (itself optional) refers to the list of files to be renamed.

[-AS] file names
: The list of new file names is (optionally) preceded by the keyword -AS.

[-BEFORE] date
: Rename the file only if its datestamp is before the date and time specified (or if it does not have a valid datestamp). See 4.6.2 for the date format.

[-AFTER] date
: Rename the file only if its datestamp is after the date and time specified (or if it does not have a valid datestamp). See 4.6.2 for the date format.

The standard arguments for utilities are also available, see 4.9.l-2.

Examples

```
-> rename oldName newName
-> rename -from this -as that -force
```

```
-> rename old new -verbose
-> rename S.adir.* -as S.bdir.* -force
-> rename adfs:*- -as adfs:*-rif -confirm
```

Functions

The rename command lets the user rename a list of files or directories. Both the original and new files must be on the same medium, i.e. same physical surface of the disc. If it is required to `rename' across filing systems or discs, use the Copy command with the -delete option which is effectively the same thing.

If wildcards are used, the matched wildcards in the source list are substituted for the corresponding wildcards in the destination (if both source and destination strings are wildcarded).

6.11 Set Command

This utility allows the user to set the value of various Panos attributes, viz date and time, tabs, global variables, vdu characteristics, function key bindings, and current working directory.

In effect it is a family of commands: Set Date, Set Tabs etc.

Concepts

See:

2.2	attributes
2.5.5	current working directory
2.7	global variables
4.4	function key bindings, tabs
4.6.3	time and date

General Form

-> Set Attribute arguments

Attributes and Arguments

DATE/TIME date
: This sets the date or date and time to the date specified by the -DATE or -TIME keyword. If no time is specified, then 00:00:00.00 is assumed; if no date is specified, then the current date (if set) is assumed. See 4.6.2 for the date format.

  The day of the week can also be submitted. The word `Today' is also allowed and is assumed to have a time of just after midnight yesterday. This is useful for backing up files which have been altered during the day.

TABS [-AT positions] [-THEN increment]
: Sets the tab stops for the tab key. The tab stops can be set to incremental or absolute positions.

Incremental positions are numbers prefixed with a plus sign, and are relative to the previous tab stop. Absolute positions are simply column numbers.

The -THEN keyword describes column intervals. For instance, -AT 4,12, + 2 -THEN 8 would set tab stops at positions 4, 12, 14, 22, 30, 38 and so on.

VAR [-NAME] name [-VALue] value
: This sets the value of a global variable. This set built-in command is similar.

VDU attribute value
: This sets the value of a VDU attribute:

-MODE	screen mode,
-COLour	text colour,
-FOREground	as colour,
-BACKground	background colour,
-PAGEd	page mode (enabled-TRUE or disabled-FALSE).

For representations of the values for each of these, see 4.6.3.

KEY key-number string
: This assigns a string to the specified soft key. Typing I j at the end of the string will cause a carriage-return to be appended to the string when the specified key is pressed.

DIR directory
: This sets the current working directory to the directory specified. The directory name may include wildcards.

The standard arguments for utilities are also available, see 4.9.l-2.

Examples

```
-> set date 15th feb 85
-> set date monday 1st august 1983 4:14 pm
-> set date 05-mar-85 12:56:00.67
-> set time 2:02 pm 1985/february/15

-> set tabs -at 0, 4, 10, +6, +6, +6 -then 8
```

```
-> set var -name cli$path -value dfs::0, dfs::2, a
-> set var cli$prompt "& "

-> set vdu -mode 3
-> set vdu -colour cyan
-> set vdu -background flashing-red
-> set vdu -paged true

-> set key 3 "edit myprogramlj"

-> set dir $
```

6.12 Show Command

This command acts in a way complementary to set, i.e. it shows the values of various Panos attributes, rather than setting them.

Concepts

See:

2.2	attributes
2.5.5	current working directory
2.7	global variables
4.4	function key bindings, tabs
4.6.3	time and date

General Form

-> **Show attribute arguments**

Arguments

*[-TO]*name
See 4.9.l.

[-SET]
Formats output ready for re-setting. A use for this is the creation of a command file which will set the shown attributes to their current values.

Attributes and Arguments

DATE/TIME
This shows the time and date in textual format.

TABS
This shows the current tab settings.

VAR [variable-name]

This shows the value of one or more global variables. If no variable-name is given then all global variables are printed. Wildcards may be used.

VDU [attribute]

This shows the value of a VDU attribute: see under Set command for a list. If no attribute is given, all are listed.

KEY [-Numbers] [number]

This shows a soft key's string. The key number is optionally specified by a -Numbers keyword, i.e. show key -numbers 3,4,6. If no keys are specified, then all are shown.

DIR

This shows the full pathname of the current working directory.

The standard arguments for utilities are also available, see 4.9.l-2.

Exarnples

```
-> show date
-> show time

-> show tabs -set

-> show var
-> show var program$*
-> show var file$* -to setfiles-cmd -set
-> show var file$-mod,alias$*

->  show vdu mode

-> show key 3,4,5,9,2
-> show key -numbers 7,2

-> show dir
```

6.13 Star Command

This utility allows the user to issue BBC style * commands.

It is a `dangerous' command which enables the user to send commands to the BBC Microcomputer's command line interpeter. Such commands are usually preceded by a'*' in environments such as BASIC, hence the command's name.

Concepts

See the *Panos Programmer's Reference Manual.*

General Form

```
-> Star arguments
```

Arguments

[Command]
This is the text of the command to be issued.

Examples

```
-> star fx 5,4
-> star screenDump
-> star free
-> star map
-> star compact
```

Functions

Since Panos does a lot of work `behind the scenes' to keep up its environment, the result of issuing * commands is not always predictable, and they should only be used when no other means exists. The integrity of Panos cannot be safeguarded.

The current directory is selected for the operation of a star command.

7 Editor

7.1 Introduction

7.1.1 Context

The editor is used in the creation and modification of source programs for the language compilers and the assembler. It may also be used for general applications, for example, preparing manuals which are formatted by a formatter program such as GOAL, or for preparing data or command files. In fact it can edit any file, text or binary.

It is screen- rather than line-orientated. That is, text is displayed a `page' at a time on the screen and may be altered by moving the cursor around using the cursor keys, and by typing.

The concept of windows is fundamental to the editor; several files can be edited simultaneously, and material can easily be transferred between files which are viewed through their own separate windows.

Access to the Command Line Interpreter is also available from the editor, so compilers, utilities and built-in commands can be used from within an editor window. For example, this is useful for inspecting and switching directories. One particularly useful feature is the ability to compile programs from within the editor whilst editing, the program source, and sending the error messages to another editing window; thus the compile-edit-recompile cycle is much smoother, easier, and less time-consuming.

Because the editor is principally designed for the preparation of source texts, it has powerful search and replace facilities which includes sophisticated pattern-matching.

Most of the editor's facilities are attached to function keys (f0 to f9 in conjunction with SHIFT and CONTROL). Generally the action of a command issued by pressing a function key is to alter the text in an editing window with immediate display of the alteration. Some commands, however, cause a 'prompt' window to appear on top of the original editing

window. This window prompts the user for a particular response, and disappears after the response has been made.

A few commands are carried out using control keys. Except during prompts, the normal printing (i.e. non-function, non-control) keys are used for inserting text into the text window. The editor is relatively 'mode-free'. In particular, there is no overtype mode.

A summary of the function of each key may be found in the key cards supplied with the system.

7.1.2 Basic Facilities

In common with similar programs, the Panos editor is best understood and learnt through use and experience. Before the remainder of this chapter, a simple demonstration and exercise are given which will serve to introduce the basic editor facilities.

Try entering the small Pascal program listed below. If preferred an equivalent C or FORTRAN 77 program, or plain text could be substituted. Program sources in various languages can be found on the Welcome disc. The confidence test (`Hello World') found on each language disc could be used as a basis for exploring the editor.

```
PROGRAM World (Output);

( Acorn 32000 ISO Pascal - basic confidence test )

BEGIN
   WriteLn ('Hello Pascal world')
END.
```

First load the editor. To edit a new file type:

```
-> edit
```

When the editor has loaded, it will announce itself with a version number and after a short pause set up the screen with a white border around a black background, i.e. an editing buffer appears which is viewed through a window occupying almost the entire screen. An end of text marker which looks like End of Text is placed at the top lefthand corner of the screen. As characters are typed, the marker moves to the right.

To enter the first line, simply type it in. As usual, letters are entered in the case determined by [CAPS LOCK]. Pascal ignores the case of all characters outside of strings, so the program above could be typed in either case.

It is important to note that the normal line editing keys (see 4.4) do not necessarily have their usual actions within the editor.

The [DELETE] key, however, is similar. A mistake typing the first line, for example, can be corrected by pressing the [DELETE] key. This key erases the character to the left of the cursor and moves the cursor left one position. Holding down [DELETE] enables several characters to be erased. Over-enthusiastically deleted text may be restored by pressing the combination [SHIFT] - [TAB]. The alternative method of deleting characters is to use the [COPY] key. This removes the character at the cursor position, shifting the rest of the characters on the line to the left. It is often easier to use [COPY] when removing a series of characters, as the cursor is placed over the next character to be deleted.

Note that this use of [COPY] is certainly non-standard, and does not copy! It is however very convenient within the editor to have the complementary delete keys next to each other on the keyboard.

Once the first line has been typed without error, press [RETURN]. This moves the cursor onto the start of the next line and inserts an invisible end of line character at the end of the first line.

The fourth line is indented slightly. To do this, press the space bar the appropriate number of times (four in the example above), or alternatively press [TAB]. This key tabulates the line into columns eight spaces wide by default (but see section 6.11). Press [CTRL] - [T] and tab spaces will show up as arrows (on the screen only, not on printed text). This is the best way of seeing how [TAB] works.

Once some text has been typed in, try editing it. Editing entails adding, deleting, moving and changing text.

To add text, simply type it in where it is required; it will be inserted at the cursor position. The cursor can be moved to any location on the screen page by using the four cursor-control or arrow keys. Pressing one of these moves the cursor one character position in the indicated direction. Note, that it is not possible to move the cursor above the top of the text or below the End of Text marker.

Experiment with the cursor keys to see the effect they have on the cursor.

Note that the editor always operates in insert mode, that is, characters typed in have the effect of 'shifting-along' any text which follows. This is distinct from overtype mode editors in which new characters replace characters already present.

To conclude this simple editing session, save the text; press (f5). This is the function key marked f5 above the row of number keys. A prompt window will be created and a request will be made for a file name. Type the desired name (e.g. test-pas) followed by (RETURN). The file will be saved under the name given and then the prompt window will be removed.

To exit the editor, press (SHIFT) – (f3) and respond 'yes' to the prompt.

Organisation of this chapter

By way of introduction, many of the basic facilities have been described through example. The concepts behind the use of the editor are documented next. This is followed by the organisation and start-up.

The editor has its own command language based on the use of function keys, so this is documented along with many of the simple facilities in a section on its own. Associated with this language is a display plus feedback, so these too are documented separately. Finally more complex user actions are defined in terms of the facilities provided.

7.2 Concepts

The editor is based on the concept of windows. An edit window is conceptually a limited sized port onto a text document. The window may be moved around to gain different views of the document. Each window has a corresponding buffer (area of memory), of a given and fixed size. Associated with the port is an area of the screen which displays the current text.

In addition to edit windows, other forms of window are used for dialogue, help etc. as follows:

prompt window (a dialogue box)
command window
error window (an alarm box)
help window

When multiple windows co-exist, they are usually mapped onto the screen in such a way that some are on the top of (and thus wholly or partially obscuring) others. However, there is always a window which is `active' at any one time, i.e. the window which contains the cursor, and upon which all functions take effect. This applies to all windows including non-editing windows such as prompt windows. In **fact such windows normally only exist when they are active.**

The cursor has two separate functions. The first indicates the current insert point for typing new text which is conceptually immediately before the cursor. Secondly it selects individual characters, lines, or blocks of text. The cursor is guaranteed to be visible if the active window is on top.

7.3 Editor Organisation

7.3.1 Installation

The editor should have been installed along with the Panos system, as described in the *User Guide* **supplied with the system. Difficulties in getting the editor to work properly are dealt with in 7.8. If using the editor in conjunction with the DFS, a system disc should have been created, which is placed in the top drive.**

A number of global variables affect installation - see 7.3.3.

7.3.2 Loading the Editor

To load the editor, enter Panos as usual. If using Panos in conjunction with the DFS, insert the system disc in the top drive (e.g. the language system disc).

A number of options can be issued which mainly affect start-up behaviour. Various global variables can also be set by the user which can further determine the editor's behaviour; these are described in 7.3.3.

General Form

The editor may be called by typing:

```
-> edit
-> edit filename options
-> edit arguments
```

The last form is just the general case where arguments may include a file name or may be empty. The second form loads a file to be edited, the first does not.

Arguments

-Help
: See 4.9.l. It prints out a list of the options. Note that it is not the same as, nor does it give access to, the help information associated with editor functions, which is obtained by pressing `SHIFT - ESCAPE` from within the editor, and is fully described in 7.6.l. This is what the `-help' option will produce (for version l.10):

```
-> edit -help
Edit 1.10
Keywords:
  -File <file>           File name to start editing
  -Line <number>         Line number to jump to
  -Buffer <size>         specify buffer size (50000 default)
  -Obey <file>           Obey KeyLog file
  -New                   Create a new file
  -Browse                Prevents altering or saving file
Global variables:
  EDIT$HostCode         Host-code file name
  EDIT$Initfile         Optional initialisation file
  EDIT$ScreenMode       Optional screenmode (0/3)
  EDIT$Extension        Optional default file extension
  EDIT$KeyHistory       Optional keystroke history file
  EDIT$HelpFile        Optional help file
```

-Identify
: See 4.9.l.

-File

The keyword -File is not normally required since positional notation is the norm for the filename.

-Buffer

This option is used to set the size of the buffer. The default size is 50,000 bytes. The maximum buffer size depends upon the memory (RAM) available. Note that when a file is first loaded into the editor, the buffer is automatically set to accommodate the size of that file.

-Line

This option specifies a line number. In the example below, when the file is loaded, the cursor will be at line 45. See also section 7.7.2 (searching) and section 7.7.4 (jumping to a line).

-Obey

This obeys a file which contains a record of all key strokes (including function keys) which have been made during a previous session with the editor. Creating and using keylog files is dealt with in 7.7.3.

-New

If this keyword is specified without a file name, then the effects are the same as typing `Edit' on its own, i.e. a new editing window is created with no name. However, if a file name is specified then when the file is saved, it will be given the specified name as a default.

-Browse

Sometimes it is useful merely to view a file with no intention of changing it at all. As a safeguard against accidents, the keyword `-Browse' permits editing without altering or saving the file.

E.raniples

```
-> Edit -File buffer
-> Edit bigfile3 -buffer 600000
-> Edit dfs::2.Mop-cmd -line 45
-> Edit Frog -New
-> Edit Precious-txt -browse
```

7.3.3 Global Variables

The behaviour of the editor is governed to some extent by the values of certain global variables, the meanings of which are described in this section. As is usual with such variables they may be set by the user (see 2.7).

In the version of !Panos supplied with the system, there are two global variables pertaining to the editor, although only `Edit$HostCode' must be set for the editor to function.

Edit$Hos tCode

This global variable must be set to the full pathname of the editor host code. In the versions of !Panos for the ADFS or NFS and for the DFS these are different, with values as follows:

AD/NFS	"$.Panoslib.Edit6502-bbc",
DFS	":0.ed6502-bbc".

The only reason why this should ever be changed, is if the pathname of the host code changes (for instance, if the name of Panoslib is altered).

Edit$ScreenMode

By default, the editor is in screen mode 0. This can be switched to 3 if desired. In mode 3, the windows have dotted line boundaries, less lines per window, and the cursor is larger and easier to detect. To change mode from 0 to 3:

```
-> set var edit$screenmode 3
```

Edit$Extension

This variable sets the default file extension for files to be edited, so, for example, if FORTRAN 77 source files were the only type of file ever edited, then it would be convenient to set this variable to `-f77'. For example:

```
-> set var Edit$Extension f77
```

This means that -f77 extensions need never be specified. However, in order to edit a file which has no extension, it would be necessary to add "-" to the end of the file name.

Edit$HelpFile

The editor has help facilities which reside in a file whose name is the value of this global variable. Values supplied with the default !Panos are:

AD/NFS	"$.PanosLib.EditHelp-dat"
DFS	":O.edHelp-dat"

This global variable need only be altered if the help file has been moved, for instance, if `Panoslib' has been renamed.

Edit$Cotntnand

This variable determines the window and buffer size of the command window. The general form for its value is:

```
"-height h -width w -buffer b"
```

where h is the height of the command window in lines, w is the width in characters, and b is the buffer size in bytes. For example:

"-height 12 -width 77 -buffer 2000"

is the default.

Edit$KeyHistory

See 7.7.3.

7.4 Display

7.4.1 Screen Layout

The screen is divided into editing windows and a top status line. The status line is used partly for instructions, (see 7.6.2), and partly to display a clock (see 7.4.5).

Pop-Up windows are superimposed temporarily onto the screen at a system defined place for dialogue, warnings etc., see 7.5.7 and 7.6.

Editing windows display a portion of the text in the buffer associated with that window. As editing takes place the display is updated. However the re-display onto the screen takes place in a way that may be unfamiliar, and a little strange at first. The screen is only re-displayed every `clock tick' and when the editor is not busy carrying out an action.

The effect is the user does not have to wait for the editor to redraw the text before continuing with the next action. Indeed if a complex sequence of actions are carried out only the final display and perhaps some intermediate ones may be seen. This has little effect on the beginning user, but speeds up operation for the advanced user.

7.4.2 The Cursor

The cursor is displayed in the `active' window as a flashing underscore, the exact form depends on the screen mode. Each window has a separate cursor, those not in the active window have a different, but non-flashing representation. Note that this applies to all windows, including for example prompt windows, i.e. when a prompt window is active the cursor is employed for the user response.

A flashing cursor indicates a modified buffer. If the buffer is newly loaded or saved, then the cursor does not flash.

The cursor is visible if the active window is on top, or if not obscured by a window that is above it.

7.4.3 Position Indicator

As the cursor changes its position, a thin vertical line in the top border of the screen moves to the right. This indicates where the cursor is in relation to the whole of the text. When this line is in the extreme left position, the cursor is at the top of the text; when it is in the extreme right, the cursor is at the end of the text. This facility operates for all editor windows.

7.4.4 Line Overspill

Lines of text are sometimes longer than the width of an editing window; this may be particularly prevalent in documents not originally created using the Panos editor and in non-text files. Line overspill is indicated by two dots cutting into the right border of the window.

For example, move the cursor to the start of a line and hold down a key_ It will start to auto-repeat after a while, and the line will fill up with that character. When the cursor has just passed the end of the line, release the key. Press the cursor left key until the cursor re-appears on the line. A white triangle appears to the right of the border. This shows that the cursor is located beyond the edge of the window. The two small dots which remain mean that there is more text on this line and act as an indication that what is on the screen is not a representation of the whole text.

There are several ways in which the text that is missing on the right of the screen may be revealed.

Firstly, CTRL - F changes the setting of the 'folded mode' switch. In this mode, characters which would normally be off the screen are shown on the next line down. The border character changes to a curved arrow to indicate that the second line is not separate from the first but a continuation of it. This mode may be disabled by pressing CTRL - F again.

A second method is to type RETURN at the end of a word just before the edge of the editing window. Characters off the edge will now form a new line. Text saved using this method retains the new lines.

The third method, is to press CTRL - SHIFT →. This shifts the whole window including the cursor to the right, displaying text which would previously have been invisible.

CTRL - SHIFT can be used with any arrow key, shifting the window in the appropriate direction. See section 7.7.5 for more details about this type of cursor movement.

7.4.5 The Clock

When the editor is idle (i.e. not performing an operation), a clock at the upper right hand side is constantly `ticking'. The clock uses the system variable sys$date, which is initialised when the date is set.

7.4.6 Display of Special Characters

The editor is able to accept and display all ASCII characters. Printing characters are displayed in edit windows in the obvious fashion.

Non-printing characters are displayed as their hexadecimal value in a small, underlined type style. Each occupies two normal character positions on the screen but is edited as if it were a single character.

For details of input of non-printing characters, see 7.5.

In addition, a special representation may be used for tabs.

7.5 Command Language and Basic Functions

Introduction

The editor is loaded in the normal fashion by the command line interpreter, see 7.3.2. Once loaded, actions are carried out entirely through the use of special keys, i.e. the editor has its own command language tailored for its particular function.

This section introduces the command language in terms of groups of special keys. These groups are:

- printing keys
- cursor movement keys
- deletion keys
- control keys
- function keys
- other keys

Some keys have different meanings outside of edit windows, i.e. in prompt windows, help windows etc. These are dealt with separately.

Tables summarising the effect of the special keys may be found in the key cards supplied with the system.

The special keys may themselves be inserted into the text by prefixing them with the 'escape' character [CTRL] - [\]. Thus [CTRL] - [\] [f0] inserts the character whose ASCII value is 16_80.

7.5.1 Printing Keys

Printing keys simply insert text (see 7.1.2). The key [RETURN] introduces a newline character NL (ASCII LF, value 10) and is displayed as a new line, but is otherwise invisible.

Highlighting Tabs

Pressing [CTRL] - [T] will show up any tabs in the text as a long right arrow. This can be useful as it is otherwise impossible to detect the difference between spaces and tabs. Pressing [CTRL] - [T] once more will cause all of the arrows to disappear.

7.5.2 Cursor Movement Keys

Pressing the cursor keys moves the cursor one position. Larger movements are available using the [SHIFT] and control keys. The effect of all cursor movement keys is shown in Table 7-1.

Table 7-1 Cursor Keys

1	moves the cursor up one line
1	moves the cursor down one line
	moves the cursor left one character
	moves the cursor right one character
SHIFT-t	moves the cursor up one page (window)
SHIFT-4	moves the cursor down one page (window)
CTRL--	moves the cursor to the start of the line
CTRL--	moves the cursor to the end of the line
CTRL- t	moves the cursor to the start of the text
CTRL-4	moves the cursor to the end of the text
CTRL-SHIFT r	displaces the window to the right over the buffer
CTRL-SHIFT -	displaces the window to the left
CTRL-SHIFT t	displaces the window upwards
CTRL-SHIFT 1	displaces the window downwards

Note: lines are regarded as sequences of characters terminated by newline characters. Thus if a line spans more than one screen line and line-wrap is enabled by CTRL - F, CTRL - ← and CTRL - → may move the cursor up or down screen lines as well. ← at the start of a line moves the cursor to the end of the previous line. However → at the end of a line effectively extends the line with spaces.

It is not possible to move the cursor before the beginning of the text, or after the end-of-text marker.

See also section 7.7.4, which shows how to make more specific movements within the text.

7.5.3 Deletion Keys

The keys COPY, DELETE, and CTRL - U may be used possibly in conjunction with CTRL to delete a single character, or all or part of a line. This is summarised in Table 7-2. In this context, newline acts as a normal character, thus DELETE at the beginning of a line, or COPY at the end will concatenate two lines.

Table 7-2 Deletion Keys

DELETE	deletes the character to the left of the cursor
COPY	deletes the character to the right of the cursor
CTRL-DELETE	deletes the line to the left of the cursor
CTRL-COPY	deletes the line to the right of the cursor
CTRL-U	deletes the whole line of text where the cursor is located

7.5.4 Control Keys

A few control keys including CTRL - U, CTRL - \, CTRL - F, and CTRL - T have particular effects. These are documented separately.

The key CTRL - M which represents the end of line character in BBC Microcomputer prepared text files is treated as a printing character, i.e. it may be inserted directly without escaping it with CTRL - \

7.5.5 Function Keys

Many of the editor's facilities are accessed by pressing the function keys, possibly in conjunction with SHIFT or CTRL. Generally, a function which requires a key to be pressed in conjunction with SHIFT is more 'powerful' or 'dangerous' than other types of function requests. For instance, f1 used on its own copies a block of text, reproducing the original at another location; SHIFT - f1 moves a block, deleting the original.

These are documented in terms of the actions they carry out in subsequent sections.

7.5.6 Other Keys

The ESCAPE key has no effect during normal editing, but abandons the current action in command windows, error windows etc.

SHIFT - TAB may be used to undo the effect of previous deletions. See 7.1.2 for details.

7.5.7 Prompt Windows (Dialogue)

A number of commands (function keys) require further input from the user, e.g. a file name or a search pattern. A prompt window (or dialogue box) appears, and the user is invited to type a response.

The prompt window is an editing window in its own right. Thus the usual cursor movement and editing keys are available (such as [DELETE]) and it is even possible to use an editing function such as searching.

However, the [COPY] key has special properties in a prompt window; instead of deleting one character at the cursor position, it repeats the last string entered in that context, be it a search/replace pattern, or a file name. This is particularly useful when saving files, as the full file name need not be retyped.

7.5.8 Command Windows (Panos CLI)

Command windows provide access to Panos commands. Press [f3] and a command window will appear containing the Panos -> prompt. For example, to catalogue a directory type:

```
-> Cat
```

To return to the editor, press [RETURN] or [ESCAPE].

Almost anything that is normally carried out by Panos (except running the editor) can be done from this command window. Not only can the utilities be run, but also the compilers, linker, and user programs.

This is very useful for debugging. A typical cycle for program development may be as follows:

1. **Edit program source**
2. **Save program source**
3. **Compile program source from within an editor command window**
4. **Scroll up and down from within this window, making notes of the errors**
5. **Return to editing the original program source**

Note that throughout the session, the program source remains visible in the main editing window. The actual commands used (for a Pascal program called `test') would be:

```
-> Edit test-pas
   {Editing commands}
(f5) - (COPY)
(f3)
-> Pascal test
(RETURN)
```

Using the compiler `-error' option, and then copying the error file to an editor command window gives a glimpse of the errors resulting from the compilation. Because the window is smaller than the potential size of the error list, only a few lines can be present in it at any one time. However it is possible to scroll up and down within this window, carry out searches and other functions just as in the main editing window.

An alternative method is to load in the error file to a separate edit window. Window creation and manipulation is explained in 7.7.5.

The key (COPY) has a special meaning in a command window: it repeats the previous command.

The size of the command window is governed by the Edit$Command global variable, (see 7.3.3).

Note that the use of command windows is subject to memory constraints.

7.6 Feedback and Errors

The editor provides feedback to the user in a number of ways. Normally this is documented under the particular event, but a number of common or special items remain to be described in this section. These consist of help information, and error messages.

The cursor may be used to select a character, or line, and in conjunction with function keys, a block. Feedback is given to indicate the selected character or block, see 7.5.2 and 7.7.l.

The position of the cursor as a proportion of the text is also displayed, see 7.4.3.

The text buffer itself is displayed through an edit window, see 7.4.l.

7.6.1 Help Information (and Windows)

There are three sources of help for the editor: this document, which offers the most comprehensive information, the `keyboard cards' which serve as an at-a-glance reminder, and on-line help. This section describes the on-line help facility, which is entered from within the editor and is viewed in a help window. This is not the same as the help information about start-up options, which is accessed outside the editor as described in 7.3.2.

Help Windows

On-line help can be accessed from within any text window by pressing the keys (SHIFT) - (ESCAPE) . It consists of a series of three 'pages', with headings 'General', 'Keys', and 'Buffers'.

There are two ways of moving around the help text: using the cursor keys to move between headings, and pressing (SHIFT) - (ESCAPE) to enter a page, and (ESCAPE) to exit a page. Note that not all the information displayed on a page represents all the detail there is: experiment to see what the on-line help does contain. A short tutorial exercise is given below.

Exercise

1. Press (SHIFT) - (ESCAPE)

2. Notice the `highlight cursor'. This draws attention to the present location within the on-line help text. At first, this is placed over `General'.

3. Press the left-arrow cursor key once. This will now change pages to the `Keys' page.

4. Now enter the 'Keys' page by pressing (SHIFT) - (ESCAPE) . A list of all of the editor function keys and the actions they perform will appear.

5. Press the down-arrow cursor key a few times, and the function key headings will be highlighted.

6. Now press the right-arrow cursor key three times, and control key functions will be displayed.

7. Press (ESCAPE) to return to the previous help page, and now explore the 'buffers' help page.

8. To return to the main editing window, press [ESCAPE] whilst in the first ('headings') help page.

7.6.2 Error Messages (and Windows)

There are two types of errors: some cause the appearance of error windows, and others generate a warning message which appears at the top left hand corner of the screen above the normal editing window.

Warning messages draw the user's attention to the fact that something has happened which may not have been intended, but which has not caused the editor, Panos, or a filing system to behave abnormally. An example of a warning message is

```
Warning: Search not found
```

This message remains until [ESCAPE] is pressed.

No user action is normally required. However the warning may be associated with some kind of 'recovery' action by the editor. In the above example it is assumed that a different search is required, so the prompt window is redisplayed. At this stage either the corrected search pattern may be typed, or [ESCAPE] pressed to terminate the action.

Error Windows

Error windows (or action boxes) appear following a more serious user ,error'. In contrast to warning messages, errors which produce an error window cannot be ignored, and until some action is taken, editing cannot be resumed. This following is an example of an error message.which occurs when an attempt has been made to load a non-existent file called faulty:

```
Error: From module File
File 'Faulty' does not exist
```

If the user now types anything other than [ESCAPE], another message appears in the window:

```
[[ Press Escape to continue ]]
```

After pressing (ESCAPE), the user is returned to the 'Load' window to have another attempt at spelling the file name.

7.7 Advanced Editor Functions

Specific editor functions are described in this section expressed in terms of actions required by the user.

7.7.1 Block Editing

Single characters or lines may be deleted using (DELETE), (COPY) or (CTRL) - (U). It is often required to delete or to move or to copy a section of text. These three functions may be performed using markered blocks.

Selecting a Block

A marker is a notional position indicator embedded in the text. It is set by moving the cursor to the desired position and pressing (f0).

The start of the markered block is indented after the sign Marker.

The end of a block may be either a second marker, or the cursor itself. If a second marker is set, the first marker is replaced by.Block Start, and the second marker is called Block End.

To delete a misplaced marker, press (f2). If two markers are set, then both are deleted when (f2) is pressed, unless the cursor is placed at one of the marker positions, in which case, only the other marker is deleted. Markers are automatically deleted after block operations, except for block copying. This is so that more than one copy of a block can easily be made.

Block Commands

The block commands are:

f0 Set a marker at the cursor position

f2	Delete marker
fl	Copy the block delimited by marker 1 (Block Start) and marker 2 (Block End) to the cursor
SHIFT-fl	Move the block of text delimited by marker 1 (Block Start) and marker 2 (Block End) to the cursor
SHIFT-f2	Deletes the block of text between the cursor and the single marker.

Block Deleting

The sequence of actions for deleting a block is:

1. Set a marker at the top of the piece of text to be deleted. Only one marker may be set during a block delete.
2. Move the cursor to just after the last character to be deleted.
3. Press [SHIFT] - [f2]. The text between the marker and the cursor will disappear. The marker itself will also be deleted.

Pressing [SHIFT] - [TAB] will restore mistakenly deleted text.

Copying a block

The sequence of actions for copying a block is:

1. Position the cursor at the top of the text to be copied and set a marker there by pressing [f0]
2. Position the cursor at the end of the text to be copied and set a marker there by pressing [f0] again
3. Move the cursor to the point in the text where the text is to be copied.
4. Press [f1]

Moving a block

The sequence of actions for moving a block is:

1. Position the cursor at the top of the text to be moved and set a marker there by pressing [f0]

2. Position the cursor at the end of the text to be moved and set a marker there by pressing (f0) again
3. Move the cursor to the point in the text where the text is to be transferred
4. Press (SHIFT) - (f1)

7.7.2 Searching and Replacing

Searching for occurrences of patterns within text, (sometimes called strings) and replacing them with other strings, is achieved using the function keys (f7) and (f8). Searching for strings is a useful technique for jumping to a particular place in a document. Using these function keys with (SHIFT) repeats the last search or replace:

f7	find a search pattern
SHIFT-f7	find the next occurrence of the last pattern
f8	replace one string by another
SHIFT-f8	replace next occurrence by the same string

When (f7) or (f8) is pressed, a prompt window appears at the edge of the main editing window, containing a request for a pattern:

```
Search C
```

or

```
Replace C ...      I by I
```

A legal search string is anticipated (i.e. one which conforms to the rules described below). After entering the string, press (RETURN) and the closing ']' will be added. Replacing is a similar activity to searching, except that a replacement pattern is also requested, and the match that has been encountered will be replaced by the replacement string.

Scope

Normally the scope of a search or replace is from the cursor to the end of file. The scope within which a search or replacement is carried out can be restricted to a block by planting one or two markers which then form boundaries beyond which all operations are inhibited providing the cursor is within the block when the command is given.

Normally searching takes place in a forwards direction. However, typing CTRL - B as the very first character in the pattern causes the search (or replacement) to be performed backwards from the cursor, stopping at the start of the file. To provide feedback, CTRL - B in this context is displayed as a left-pointing arrow.

Search Patterns

Simple patterns consist of plain text which the editor looks for exactly as it has been typed (with the exception that upper and lower case letters are treated as equal). For example, the following sequence of events will look for the string `procedure':

1. **Move the cursor to the start of text area to be searched (normally the start of the file).**
2. Press f7 - search for a string.
3. Type the pattern 'procedure', followed by RETURN.

If any occurrence of the pattern is found, the cursor will be placed at the start of it, otherwise the cursor position is unaltered, and a warning message will be displayed at the top left-hand corner of the screen just above the main editor window:

```
Warning: Search not found
```

This message disappears when re-entering the main window by pressing ESCAPE. To find subsequent matches of 'procedure', press SHIFT - f7. This can be repeated until the last match has been found.

In addition to simple strings like `procedure', patterns may contain special characters which will match more general characters or groups of characters. A list of these special characters is given in Table 7-3.

Sometimes it is necessary for these special characters to be used as literals, e.g. it may be required to search for the string "$". In such a case it is necessary to prefix the special character with the pattern escape character \. Non-printing characters in a search or replacement pattern should be prefixed as usual with the (insert) escape character, CTRL - \.

To provide feedback that a special character has been typed, it is displayed in [redacted] in the prompt window. So, for example, the keys pressed to specify a search pattern which matches the digit '1' followed by some

other number, would be 1 p. The `#' symbol would be displayed in inverse video as shown. Literal uses of these characters are not displayed in inverse video.

Special characters can be used in many combinations and there are few restrictions on the number of one particular pattern that can form a search pattern. So, for example, the search pattern the " will match any number of Vs, h's, and e's in succession.

The [COPY] key has a special meaning within the search or replace prompt window. It means the previous search, or previous replacement string.

Table 7-3 Special Search Characters

.	matches any character, including spaces, punctuation marks etc.
$	matches the newline character
@	matches the 'identifier' characters 0-9, a-z, A-Z and @
#	matches the digits 0-9
* c	matches zero or more of c in sequence, i.e. the null string, c, cc, ccc etc. This always finds the shortest match
^ c	matches one or more of c in succession, i.e. c, cc, ccc, cccc etc. Always finds the longest match
c1 - c2	matches any single character between c1 and c2 in ASCII order
[abc]	matches any one of a, b OR c, where a, b, and c are any single characters
! c	matches CTRL-c if c is in the range @ to _ (ASCII 64 to 95). CTRL codes may also be typed directly. If c is ? then !? stands for DEL (ASCII 127)
!! c	matches the character with ASCII code of c plus 128
£	is a switch for case sensitivity. Normally upper and lower case in patterns and text are treated as the same, so cAt will find cat, CAT, CaT etc. However, if a pound sign is given in a pattern, case sensitivity will be toggled (flipped) from the pound sign onwards, so that cAt will match only cAt, and c AT will match CAT or cAT.

Replacement Patterns

Replacing text is performed using (f8). This requires two pieces of information: the search string, which is described above, and a replacement string. For example, to replace all four-digit sequences in a file with the same four digits placed in angled brackets (thus 1234 becomes <1234>), the following is required:

1. Press (f8)
2. Type in the pattern: #### followed by (RETURN)
3. Type the replacement string: <&> followed by (RETURN)

`####' stands for any four digits, and `&' stands for a repetition of the whole search pattern.

When a match for the pattern is found, the editor will prompt with:

```
R(eplace), S(kip), 0(nce), A(ll), E(scape) or H(elp) ?
```

Possible responses and their corresponding actions are shown in Table 7-4. Only a single letter with no (RETURN) should be typed.

Replacement strings may contain special characters. These are shown in Table 7-5. As with search patterns, the special replacement characters are highlighted in inverse video. Use of these characters as literals requires them to be prefixed with the pattern escape character (\)

Table 7-4 Replacement Options

O	means make the replacement once and then stop,
E	returns to the main editing window, doing nothing,
R	makes the replacement and looks for the next match,
S	looks for the next match without making the replacement,
A	replaces all matches without prompting,
H	gives help on the command.

In dw case of O, R, S, and A, the cursor is left at the position of the **last replacement; E and H do not alter the cursor's position.**

Table 7-5 Special Replacement Characters

$	stands for the newline character
&	stands for whole of the matched string
! c	as patterns
!! c	as patterns
%n	stands for the nth (counting from zero) wildcard section in the pattern. This is best explained by illustration. Take for example the pattern a*.@. This represents the largest number of the character 'a' in succession, followed by the next character, followed in turn by an identifier character. %0 is the text matched by a, %1 is the text matched by *., and %2 represents the text matched by @.

In other words, % accesses one particular wildcarded section in the search pattern.

To save counting, %n can be used to stand for the nth match, %*n to stand for the nth * match and so on.

% + or -n is similar to the above replacement pattern but forces the matched pattern to be replaced in upper (+) or lower (-) case. Thus % + *0 will replace whatever `*'represents in upper case, and % + & will change to upper case all the characters in the search string.

\n allows literal digits to be inserted after a %n. Thus, %#3\11 will NOT replace the 311 th occurrence of the # match, but will replace the third occurrence, followed by the number eleven.

Examples of Replacements

Replace [Keith] by [Ben]
will replace all occurrences of "Keith" with "Ben"

Replace [CTRL - M] by [$]
will replace all carriage returns, (ASCII CR, value 13) with newlines, (ASCII LF, value 10). This is a way of converting BBC Microcomputer text files to Panos (and ISO) standard (or vice-versa). The CTRL - M will be displayed as hexadecimal 0D.

Replace [★[A-Z]ss] by [&$]
will replace any occurrence of upper case characters from A to Z followed by two 's's, with the matched pattern followed by a new line.

Replace [$\. @##] by [%@0$%#0 123 $]
will replace a literal full stop (.) followed by an 'identifier' character (@), and two numbers (##), occurring at the beginning of a line ($), by the 'identifier' character (%@0), a newline ($), the first of the two digits in the search pattern (%#0), and the numbers 1,2,3, followed by a new line ($).

Replace [CTRL-B ^ The *@ $] by []
will delete backwards (from the cursor to the start of the text or marker) the exact word 'The' (exact case), then any number of identifier characters until the end of the line. Replacing by nothing is equivalent to deleting.

7.7.3 Learnt Sequences

The editor has the ability to 'remember' a sequence of commands and execute them all later. The sequence of commands which carry out a number of other commands is sometimes referred to as a macro. To compose an editing macro, follow these steps:

- Press (f4) to enter 'learning' mode
- Type in text and commands as usual. These will be remembered, in addition to being executed immediately. Anything which would normally be entered at the keyboard may be typed
- Press (f4) again to terminate learning mode
- To repeat the command sequence, press (SHIFT) - (f4)

Oht-ed Sequences

It may be convenient to keep a record of actions within the editor, and possibly 'replay' them at a later stage. This is useful if the same pattern of editing is to be repeated on a selection of files. This can be done by setting the global variable 'Edit$KeyHistory' to any file name, carrying out the editing sequence, exiting the editor, and then using the '-Obey' keyword to obey the key history file. This is analogous to using Panos command files to

carry out a series of operations. The sequence is set out in the example below:

Step 1:

```
-> set var Edit$KeyHistory LogFile
```

Step 2:

```
-> Edit filet
```

Step 3: Carry out sequence of editing commands etc. and then exit.

Step 4:

```
-> rename logfile OLdLog
```

Step 5:

```
-> Edit file2 -Obey LogFile
```

Whatever actions were performed during step 2 will be repeated upon the same file in step 4. Compare this with using the learning function; the main difference is that the `logfile' is a permanent file, whereas the learnt sequence is lost upon exiting the editor, or erased when a new learning sequence is initiated.

7.7.4 Moving the Cursor

There are five ways of moving the cursor, and hence selecting a position, or defining the insert point. These are shown in the following list. The first three are described in full elsewhere, but repeated here for completeness.

1. By means of the cursor movement keys, (see 7.5.2).
2. Searching for a particular character, or group of characters, (see 7.7.2).
3. Entering the editor at a particular line through use of the `-line' startup option, (see 7.3.2).
4. Jumping to a marker. After setting a marker, press CTRL - f0 and the cursor will be placed at the marker position.
5. Jumping to a line. This can be done after or before the current cursor position by pressing SHIFT - f0. A prompt window then requests a line number. If the given line number is greater than the number of lines in the document, then the cursor is placed at the end of the text.

7.7.5 Windows and Buffers

Windows have been introduced in previous sections. In particular, prompt windows, help windows, command windows and error windows have been described in full, but edit windows have only been introduced. Some of the concepts behind windows are put forward in 7.2.

It is possible to create more than one edit window containing separate texts entirely; this enables more than one file to be edited at the same time. Furthermore, one text may be split into a number of different windows, which can then be edited and saved as separate files.

The window may be moved around to gain different views of the document, using a k - SHIFT + arrow combination.

Each window has a corresponding buffer (area of memory), of a given and fixed size. This is determined by the size of the file which was loaded. If a new file is to be edited, then the default size is 50,000 bytes, but this may be changed using the `-buffer' start-up option.

The number of windows that can be opened simultaneously depends upon the amount of memory available in the computer, and the size of the buffers. For instance, if the buffer size is set to 700,000, then a machine with one megabyte of memory will not support buffer duplication (i.e. a second window will not be supported, and an error message will be given).

When multiple windows co-exist, they are usually mapped onto the screen in such a way that some are on the top of (and thus wholly or partially obscuring) others. This has already been seen in the context of non edit windows, e.g. prompt windows. However, there is always a window which is 'active' at any one time, i.e. the window which contains the cursor, and upon which all functions take effect.

Window manipulation is carried out using a selection of keys described in Table 7-6.

All of the normal editing commands can be carried out in any window; the editing commands can also be used to transfer data between windows. For example to move or copy a block, first select the block by setting markers, then select or switch to another window and execute the move or copy command.

Note that markers can only be deleted in the active window.

Each window also has its own associated on-line help information, obtained via the help command [SHIFT] - [ESCAPE], which gives the status of the window's buffer (name of file in buffer, maximum buffer size, percentage of buffer used, etc.).

Table 7-6 Window Manipulation Commands

CTRL-f6 Create new window
: This is similar to other [f6] commands in that a prompt window appears which expects a file name to be supplied. The new file is then loaded into a window, and this new window becomes the 'active' window. There are a number of options that may follow the file name.

CTRL-SHIFT-D Duplicate a window
: This creates a whole new window (and corresponding buffer) identical in size to the original window, thereby completely obscuring it, and places the cursor in this new empty window ready for editing.

CTRL-SHIFT-S Splitting a window
: Whereas [CRTL] - [SHIFT] - [D] causes the original buffer to be copied, [CTRL] - [SHIFT] - [S] splits the current window in two at the cursor position, making a new window. The size of the buffer which the new window looks into is the same size as the original. The new window is empty to begin with, and the original window is still visible.

CTRL-SHIFT-K Kill a window
: This command deletes a window and its buffer. Be careful when using this command to delete the correct window: the active window which contains the cursor is deleted, and the cursor may not always be visible on the screen.

CTRL-SHIFT-F Extend a window
: This command extends a window to its full size, e.g. after splitting.

CTRL-SHIFT- A Select a window
: This command moves between windows, placing the next window on top of the current window. The new window is made the active one.

CTRL-SHIFT-\ Switching between windows
: This command is similar to [CTRL] - [SHIFT] - [A], but the selected window is not placed on top of the current window, although the new window is made the active one.

Exercise

As a demonstration of the usefulness of multiple editing windows, try this exercise:

1. Edit a program source file which contains errors

2. Press (f3)

3. Compile this faulty program source (from within the Panos command line window), sending the output to a named file

4. Return to the main editing window

5. Press (CTRL) - (SHIFT) - (S)

6. Load the error file into this second window

Now it is possible to switch between the two windows, using (CTRL) - (SHIFT) - (\) scrolling through the one and editing the source program in the other.

7.8 Problems

This section deals with difficulties encountered in loading and using the editor. It also covers avoiding and repairing some basic mistakes. The editor has powerful recovery mechanisms which reduce the possibility of losing edited text. Error messages and guidance in recovery are provided on-line. However, for completeness these are also documented below.

Messages in Error Windows

See 7.6.2.

PrescinY the Wrong Key

Hitting the (ESCAPE) key aborts the current activity whether running or waiting for input. This is useful if a function key has been pressed accidentally, or if an incorrect search/replace pattern has been typed in. For

example, if an incorrect search pattern has been given and the editor is laboriously churning through the file, pressing ESCAPE will halt the process.

The ESCAPE key will also terminate a command that has displayed a prompt window, and return to the edit window.

Un-deletink

If text has been deleted by mistake, then press SHI F7 -TAB and the deleted text will reappear. This also works for blocks of deleted text.

The Editor Window does not Appear.

This refers to the situation when the edit command has been given, but the screen remains blank, even after a suitable pause.

If floppy discs are being used, check that the disc containing the editor is placed in the top drive.

There is a chance that the Panos variables for locating the various parts of the editor may not have been set up correctly. If this is the case, then either the installation procedure has not worked correctly, or a customised (or corrupt) version of !Panos is being used.

First try using an original version of !Panos to start-up Patios. There are two variables which must contain the locations of various parts of the editor: `Edit$HostCode', and `Edit$HelpFile'. These are initialised by the !Panos start-up file provided with the system, and are therefore automatically initialised when Panos is entered.

However, if Patios has been started up using a corrupt !Panos file (for example, because it has been deliberately modified), and the editor variables are not present and correct, an error message results from attempting to use the editor, and the window may not appear. Check with an original version of !Panos to see what these variables ought to be initialised to, and check also that the help file and host code do exist as they are supposed to, see 7.3.1.

If it transpires that the editor has not been installed correctly (i.e. the host code or help file is not in the right place), then it is likely that other parts of Panos are also incorrectly installed; therefore, Panos should be re-installed according to the instructions provided in the *User Guide*.

Saving Files

If an illegal file name has been given, an error message window will be created. Pressing (ESCAPE) returns to the 'Save' window for another try. Attempting to save a file in a full directory will also cause an error:

```
Error : From Module BBC
Dir full : 'file name'
```

When saving files on floppy disc, there may be a shortage of space. Compacting the disc may relieve this problem.

Pie Buffer

The editor buffer is 50,000 bytes by default when editing a new file. The name, size, and percentage used of the buffer, number of lines, and line number of the cursor position can be found out by looking in the on-line help information under the title `Buffers'.

The buffer size can be set by the `-buffer' option which can be issued on loading the editor. See 7.7.5 for a description of buffers. If the buffer size has been set to be very large e.g. 700,000 bytes (using the '-buffer'start-up option), then error messages will result if an attempt may be made to duplicate the buffer, as there is no memory space left.

Energenev Exit

In the event of a disaster (software fails to perform, for instance), and it is necessary to leave the editor and return to Panos, an emergency exit can be engineered which preserves the contents of the buffer. The editing done since the last 'Save' is therefore not lost. This may be carried out at any time and from within any type of editing window.

Press (SHIFT) - (CTRL) - (ESCAPE) TWICE, and through the text, the following phrase will be displayed:

```
Sorry, But the Editor has stopped abnormally
Do you wish the buffers to be saved?
```

This awaits a 'y' or 'n' response. To return to Panos, type 'n'; to preserve, type 'y'. The editor then prompts for another affirmation:

```
Dump "<file name>" as 'BeforeA' and 'AfterA'
```

Reply `y'. If more than one buffer was open, then more prompts would appear for 'BeforeB' and 'AfterB' and so on. The initial stands for the cursor position in the document; therefore, the contents of the buffer are **saved as two files, one** which contains all the text which existed before the cursor, and one which contains text after the cursor up to the end of the file.

When the Panos -> prompt reappears, simply copy the files `BeforeA' and `AfterA' to a file name (remembering to **use** the `-force' option if the original file name is chosen), and an intact copy of the document which was being edited when the `crash' occurred is recreated.

Note that SHIFT - f5 is a much more convenient method of saving files!

8 Linker

8.1 Introduction

8.1.1 Context

This chapter describes the Acorn 32000 Linker. The linker is a utility program which runs under the Panos operating system, and is used to combine compiled object files with object libraries to produce an executable program image file, for execution under Panos.

8.1.2 Basic Functions

The command line necessary to invoke the linker begins with the word link and is followed by various arguments. In a typical example, the linker must be informed of the names of the AOF files (see later) to be linked and the names of the libraries to be used. By default, the name of the image (RIF) file to be produced is derived from the name of the first object file specified. This behaviour may be over-ridden by explicitly providing a name for the image file, following the keyword -image (described in more detail later).

Suppose a program is written in FORTRAN 77 and has two components, held in source form in the files front-f77 and back-f77. These have been separately compiled to produce the files front-aof and back-aof, and are to be linked with the FORTRAN 77 library. It is desired to call the resulting image file Test-rif. The command necessary to perform this operation is:

```
-> link front,back f77 -image Test
```

The linker will look for files front-aof and back-aof, and will link them with the library identified by `f77', producing (if there were no errors) an image file called Test-rif. (The linker's exact treatment of the argument T77' is explained in the section on libraries below.) Note that the linker supplies the appropriate extensions (-aof, -lib, -rif) if the user does not state them explicitly.

In common with all commands, lists containing more than one element have the elements separated by commas, and parts of the command are separated by spaces.

If no image file name is mentioned, the first name in the aof file list is used, with its own extension (normally -aof) removed, and -rif appended instead; i.e. in the example, it would be called front-rif, if the -image option had not been given.

A number of further options are available; these are described later.

8.1.3 Conventions in this Chapter

In terms of syntax, the basic command line of the previous section could be written as:

```
LINK aof file list (f-LIBRARY) library list) (-IMAGE file name)
```

In this chapter, parts of the command which are optional are shown in braces, (and l. Parts given in upper-case are literal text, i.e. they correspond to items which should be supplied as shown; parts in lower case correspond to general classes of items of which the user must supply a specific value. For instance, the keyword -IMAGE should be supplied (if required) exactly as it appears in the specification (although the case of letters in the actual command is not significant), whereas for the item `file name', the user should supply an actual file name in this position.

8.1.4 Organisation of this Chapter

The basic functions have already been introduced. For many users, these will be sufficient.Advanced functions are described later. Before that are the concepts behind linking, and details of the linker organisation and start-up. The linker uses the standard command language, and is installed as part of Panos, so no special treatment is required for these topics.

8.2 Concepts

8.2.1 Linking Model

The normal sequence of program development using a compiled language is:

Stage	Action	Utility	Generates (e.g.)
1	Prepare the source	editor	prog-pas
2	Compile the program	compiler	prog-aof
3	Link with libraries	linker	prog-rif
4	Run the program image		

The third stage, linking, is required in order that references to objects defined outside the main program unit may be resolved, i.e. all the procedures, functions and data structures provided by the run-time system, user libraries and Panos which are made use of, either directly or indirectly, by the program. These will normally include language-specific procedures for such things as input/output, storage allocation etc.

In addition, some languages (e.g. C, FORTRAN 77) permit the separate compilation of sections of a user's program, and the individual modules so generated also need to be combined in the same process to form a complete program. Each object referenced from the main program will reside in a module which may itself refer to other objects, and so on, so the linker has to perform a complete analysis of all these cross-references in order to determine which modules are required in the final image.

Different languages have various ways of declaring items as being external. For example, extended Pascal provides `IMPORT' and `EXPORT' qualifiers, and the `extern' specifier is used in C. In some cases a reference is implicit - for example, a W r i t e l n statement in Pascal may reference a number of routines in the Pascal library to perform output. This illustrates that even programs which don't explicitly import items have to be linked. In fact all programs will normally make reference to some external facility provided by Panos, but even those few which do not still have to be linked in order to produce a program image in the correct format for running under Panos.

8.2.2 Linking under Panos

The compilers and assembler provided with Acorn 32000 products generate output files in what is known as Acorn Object Format (AOF). This is a standard form of representation which includes (amongst other things) information relating to external references and definitions, and descriptions of the contents of code and data areas.

Files in AOF cannot be executed themselves, since as described above, external references have to be resolved, and program images loaded and run by Panos are in a different format designed for this purpose.

The action of the Acorn linker is to take input from AOF files (which normally have the extension `-aof) and library files (extension `-lib') and resolve all the references. All of the modules required in the final image are combined together, and the result is a file in Relocatable Image Format (RIF) (extension `-rif) which may be executed under Panos.

Of the language systems supplied with 32000 system, four (ISO Pascal, C, FORTRAN 77 and the Acorn 32000 assembler) require the use of the linker. In fact the assembler may be used to produced absolute or relocatable code which does not have to be linked if it is to be run directly under Pandora rather than Patios; usually though, the assembler is used to implement small efficient procedures which can be called by time-critical sections of programs written in, say, Pascal or FORTRAN 77.

8.3 Linker Organisation

8.3.1 Installation

The linker is supplied with Panos. It should have been installed along with the Patios system during the installation procedures described in the User Guide supplied with the system.

8.3.2 Global Variables

A number of global variables affect the behaviour of the link command. These are generally set during the execution of the !Panos command file, in which suitable default values are used. However, the user may wish to customise the environment, for example to a FORTRAN only system.

The effect of the individual global variables are described elsewhere in this chapter, but are listed below for reference:

Link$Lib	8.5.2
Link$Lib_List	8.5.2

8.4 Command Language

The simplified version of the link command given in the first section is sufficient for many purposes. However, there are several other options which may come in useful.

8.4.1 General Form

The full syntax of the command is:

```
LINK [(-OBJECT)      aof file list(-aof))      [8.5.17
      [(-LIBRARY)    library list(-lib))       [8.5.27
      (-FORCE        aof file list(-aof))      [8.5.37
      [-IMAGE        image file name(-rif))    [8.5.51
      |-VIA          control file(-lnk))       [8.5.41
      |-ISHORT)MAP   (map file name(-map)))    [8.5.61
      (-ABSOLUTE)                              [8•5•7]
      [-BASE         address)                  [8.5.87
      |-NOTRANSLIB)                            [8.5.2]
      (-NOLIBLISTI                             [8.5.21
      [-IDENTIFY)                              [8.4.27
      (-HELP)                                  [8.4.27
      (-ERROR        error stream)             [8.4.27
```

Again, braces, t and 1 surround optional items. Text shown in upper-case denotes literal items to be supplied, and lower-case words denote classes of object. Default file extensions are given in parentheses.

Arguments

[-IDENTIFY]
Displays version identification information. See 4.9.l.

[-HELP]
Displays a list of options. See 4.9.l.

[-ERROR]
Redirects error output. See 8.6 and 4.9.l.

The remaining arguments are described later in this chapter. In the above general form, numbers in brackets I I identify the section which explains the corresponding option.

8.4.2 Examples

Some examples of link commands (explained in full later) are:

```
-> link MyProg
-> link fProg f77
-> link fGrapb f77,PlotLib
-> link cProg c,pas
-> link MyProg,Mysub1,MySub2 f77,PlotLib,MyLib
-> link -via Sort
-> link gentest pas -image gent
```

8.5 Advanced Functions

8.5.1 Object Files

At least one file specified as an object file (or forced file, see section 8.5.3) is always required by the linker. Normally a list of one or more object files is provided on the command line (or in a control file, see section 8.5.4). The list may optionally be preceded by the keyword -object. Any file name given

which does not include an explicit extension is assumed to have the extension -aof. The list may be specified using exact pathnames and/or wild-card specifications. The linker will process the files in the order given: for wild-card expansions the order is determined by the facility in Panos responsible for this, but typically it will be alphabetical for files in the same directory.

Examples of values of the aof file list argument are:

fred	**single file**
fred, jim, nfs:stage2.sheila	**list**
:2.FPSUB??	**wild-card specification**
main,sl,s2,s3?, adfs:$.sim.f*, adfs:$.sim2.fx*	compound

Module Loading

A module contained in an object file specified as a simple object file (or in a library file which the linker requires to search) will only be included in the final image if it defines either the main entry point or a global object (i.e. a symbol or common area) required to satisfy a reference from the main program or another required module. See section 8.5.3 concerning the ability to force all modules in a **file to be loaded, whether or not they are logically required in the final image.**

8.5.2 Library Files

The second list of names, which is optional and may if desired be preceded by the keyword -library, identifies the libraries to be searched in the linking operation. Most compiled languages have their own libraries which must be linked with a program produced by the corresponding compiler. In addition, the user may have a personal library of standard routines, and the Panos library exists to provide the basic facilities such as 1/O and store allocation which all the language libraries make reference to. Procedures in the Panos library may also be referred to directly by user programs, provided that the language concerned permits this.

The syntax of the list of library names is similar to that for the list of object files described in section 8.5.l, i.e. **files may be identified by simple pathnames, by wild-card specifications, or a comma-separated list of any**

combination of these; the default extension applied to a library file name is -lib. All files so specified must exist, even if they are not actually required to complete the link operation.

Symbolic Naming

Symbolic naming may be used to automate the command. In addition to accepting file name specifications in the library list, the linker provides a mechanism (which may be disabled see later) whereby a library may be given a symbolic name: if when a library name is encountered, it is a simple name (i.e. a name with no directory, drive or filing-system components), and no explicit extension has been provided, then the linker will first check if a library of that name (with -lib appended) exists in the current directory. If one does then that file will be used. Otherwise the linker will check for the existence of a global string whose name is of the form ` Link$Lib: <name >', where < name > is the simple library name provided. If there is no such string then an error is generated; if there is one then its value is interpreted as identifying a library file or group of library files.

The permitted syntax of the string value is the same as the syntax of a list of library files (as above) except that no attempt is made to translate a simple name via the symbolic name mechanism, and hence a symbolic name may not be recursively defined. The main purpose of symbolic library naming is to permit easy reference to frequently used libraries (e.g. the language libraries for each compiled language) - it is only necessary to remember the symbolic name of such a library, rather than its full pathname, in order to include it in the linking operation.

Examples of values which might be set up for symbolic names (typically in the !Panos initialisation command file) are:

```
$ set var Link$Lib:pas          "$.PanosLib.pas"
$ set var Link$Lib:f77          "DFS::2.f77"
$ set var Link$Lib:BBCSound     ":2.sound1,:3.sound2"
$ set var Link$Lib:graphics     "NFS:$.PanosUtiLs.Graphics"
```

These would then be used by referring, in the context of a library list, to `pas', `f77', `BBCSound', and `graphics'.

Standard Libraries

In addition to any libraries given on the command line or in a control file, the linker will look for a list of libraries specified in the global string `Llnk$Lib-list' (unless this function has been disabled - see later). This enables the programmer to specify, in advance, a standard set of libraries to be searched, avoiding the necessity of including them in the command line each time the linker is used. For example, executing the command:

```
set var link$lib_list '$.PanosLib.Panos,pas'
```

will cause the library file $.PanosLib.Panos-lib, and the library identified by `pas' (typically a symbolic name) to be used automatically during each link. The syntax of the value of this global string is exactly the same as for the library list argument on the command line, i.e. a list of one or more symbolic names, pathnames or wild-card specifications, with multiple list elements separated by commas. The linker checks for the existence of this variable, and any files named in it, after it has looked for all other libraries explicitly named in the linking operation.

The order of search is important when using libraries, as once a symbolic reference is satisfied from one library, it will not be searched for in another library. The order is: first any libraries specified in a -via file are searched, then any libraries given on the command line, and finally the link$lib-list global string (if it has been defined) is used. In each case the library order is that in which the libraries occur in the list, and for wild-card specifications the order is determined by the Panos wild-card filename expansion procedures.

Disabling Symbolic Library Names

It may be desirable in some circumstances to prevent the linker from attempting to treat simple library names as symbolic ones. If this is the case then including the state keyword -nolibtrans on the command line will have the required effect, i.e. the linker will assume that all names given in the context of a library list are actual file names.

Disabling Standard Library Searches

It may be required that, for whatever reason, the list of standard libraries named in the global string Link$Lib-Jist should not be searched by the linker. It is possible to achieve this effect, by including the state keyword -noliblist in the link command line; this is obviously preferable to the alternative method which involves deleting the global string altogether.

8.5.3 Forced Files

A given input module will only be included in the image file if it is necessary to load that module to satisfy a symbolic reference of some sort. If it is desired to ensure that all modules in a file or set of files are to be loaded, the keyword -force may be given on the command line, followed by a list of object files. All modules in the specified files will be included in the image file. The permitted syntax of this (aof file list) argument is identical to that for ordinary object files (see section 8.5.l). This facility is provided to handle the situation where a particular program or language system uses a non-standard internal linkage mechanism which does not involve direct symbolic references.

8.5.4 Control File

In some circumstances a large number of files may require to be linked together, such that it may not be convenient (or even possible) to enter all of the names on a single command line. This could arise if, for reasons of modularity and maintainability, a large program is being developed which is built out of a number of small, separately compiled units. In this case it would be tedious and error-prone to have to type all the object file names at every linking operation in the development process.

To eliminate this kind of problem, a facility is provided whereby a file may be prepared which contains the names of the files to be linked together, and the linker will read the names from this file rather than requiring them all to be present on the command line. The file is in a straightforward textual format, and may be prepared by the use of the standard Panos editor. The syntax of the file contents is given as:

((-OBJECT) aof file list(-aof))
(-FORCE aof file list(-aof))
1f-LIBRARY) lib file list(-lib))

The actual contents of the file may be split across a number of lines separated by the standard newline character NL (ASCII LF, value 10); provided that the break does not occur within a file-name or keyword. The newline character is treated as a space for purposes of parsing the file. The syntax of the file list arguments is exactly the same as for the equivalent arguments on the command line itself, as detailed in the corresponding sections below.

Note that if the -VIA option is given on the command line, it is not necessary for any object or library file names to appear on it, but if they do then they will be processed after the file lists given within the control file, i.e. each complete set of object files (ordinary and/or forced object files, and library files) will be made up of the appropriate arguments from the control file followed by any corresponding arguments from the command line. The order of processing of the two sets however remains the same, i.e. all object files are processed before any library files.

An example of a link command using this option might be:

-> link -via Sort

where the file Sort-Ink contains the following lines:

Sort, SortSubl, SortSub2, IO.Forms, IO.Block, IO.VDUControl,
FileOutput,
FileInput, Verify, Compare
-Library Pascal, DBLib?

8.5.5 Image File

The final output from the linker is a relocatable image format (RIF) file. If the -image keyword is not present in the command line, the output will be placed in < obj 1 > -rif, where < obj 1 > is the name of the first file specified as an object file (or a forced file if there are no ordinary object files). Alternatively, the user may explicitly supply a name to be used by preceding it with '-image'. For example

```
-> link gentest pas -image gent
```

will place the image file in gen2-rif The default extension -rif will be used unless the supplied name includes an extension.

8.5.6 Producing a Link Map

The linker can be made to produce a map of the image file. This is a textual (read/printable) file containing details of the internal structure of the image, including the values of the global symbols defined within it. Two types of map are available, a full one which gives details of the various store areas, modules and symbols used, and a shorter version which omits the area and module information.

A map is produced by including one of the keywords -map or -shortmap on the command line. If this is followed by the name of a file or device, the map output is sent there. (For file output, the extension -map is added to the name unless it already has an extension.) Otherwise it is sent to < image > -map, where < image > is the name of the image file without its -rif extension.

8.5.7 Absolute Images

As already mentioned, the normal output file of the linker is a file in Relocatable Image Format. This is suitable for loading and execution of the file under the Panos operating system. A feature of this format is that a RIF file may be loaded and run at any address. Further, references to Panos facilities may be made symbolically, to be resolved at load time. This ensures that a program developed under one configuration of Panos will run under another.

The .alternative is to make the linker produce an absolute file (with the extension -abs) which will load and run at one address only. Such a file is suitable for execution under Pandora, not Panos, and no reference may be made to Panos facilities. This is likely to be practicable only with assembler language code.

An absolute output file is created by giving the keyword -absolute in the command line. By default, the loading and execution address of the -abs file is 16-00, suitable for loading under Pandora. The keyword -base (described in section 8.5.8) may be used to set a different address.

8.5.8 Base Address Specification

As mentioned in section 8.5.7, it is possible for the linker to produce an absolute format image rather than a relocatable one. In this case the keyword -base may be used to set the absolute base address for such a file; if it is supplied then absolute mode is assumed (i.e. it is as if the keyword -absolute had been specified). The keyword -base should be followed by a cardinal number, eg 1024, 16-.000, which is the desired address where the image should be loaded: the linker structures the image so that this is also the execution address.

8.6 Feedback and Errors

8.6.1 Redirecting Error Messages

If the linker fails to resolve all the required references, or for any other reason does not successfully complete the linking operation, error messages will be produced. In common with other Panos utilities (see 4.9.1) these are normally sent to the special stream `error:' (the screen by default), but by giving the keyword -error followed by the name of a file or device (e.g. printer:), they may be redirected to the named destination.

Note that error messages relating to the non-existence of files, and any other problems associated with the command line arguments themselves will not be re-directed in this way; the mechanism is principally of use in the diagnosis of problems to do with global symbols, e.g. unsatisfied references and multiple definitions.

9 Problems

This brief chapter lists a number of common problems in using Patios, that may confront the user, and suggests possible remedial action.

See also section 7.8 which describes editor-specific problems, and the *User Guide* supplied with the system which describes problems relating to hardware, and to installation.

Open Files

If a message of the form:

```
File already open
```

appears, particularly if it has been necessary to make an emergency exit to a program (e.g. by switching the machine off), then it is possible that one or more files have been left open. Type:

```
-> star close
```

Disc Full

If a message is displayed stating that the disc (Floppy or Winchester) is full, try deleting unwanted files. Particularly with floppy discs, it may be necessary to compact (i.e. move parts of the disc into a contiguous area). This is achieved in Panos by ensuring that the drive to be compacted is the current working drive, and then using a star command:

```
-> set dir dfs::1
-> star compact
```

BBC Text Files

BBC Microcomputer text files use a different (and non-standard) end of line character. If it is desired to transfer text files to or from a BBC Microcomputer, these characters should be converted. See 7.7.2 for one solution.

Trouble Reading Floppy Discs

There are several different floppy disc formats available for the BBC Microcomputer, and Acorn Cambridge Workstations. In the former case these include 40 or 80 track, single or double sided, single (FM) or double density (MFM), and DFS or ADFS. Discs of one format will not be read if a different format is expected. This is a potential cause of trouble.

Floppy Disc Performance

If files appear to be read or written slower than might be expected from or to floppy disc, or alternatively the disc does not operate at all, it may be that the drive parameters are incorrectly configured. Panos is supplied with the !Config file set for fast speed drives, although a slow version is provided also. See the Configure command for details.

Trouble Printing

If no output appears on a printer although data have been sent to device printer:, it may be that the printer is not configured properly. See the Configure command for details.

Trouble Porting Programs

There are of course many possible difficulties. One frequently occurring one is simply caused by the incorrect use of options with the compilers. For example, the Pascal compiler accepts strict ISO Pascal only by default. If language extensions are required, the -extend option must be used.

Trouble Configuring on Econet

The speed of the network may be too high for this application. Alternatively too many users may overload the network. The network traffic will be very different with this application than for others using BBC Microcomputers where program size is much smaller. Contact your supplier for advice.

Lost Programs or Data

Computers are generally very reliable, and users are generally very careful. Nevertheless, important files are sometimes lost through accident. You have been warned! Keep backups!

Appendix A

Table of Editor Character Codes

The table represents the mapping between keystrokes and character codes. To insert a given code, type CTRL - \ followed by the key combination for that code.

Abbreviations

RET	RETURN
SPC	SPACE
S+L	SHIFT LOCK
DEL	DEL TE
CPI'	COPY
LFT	Left arrow
RHT	Right arrow
DWN	Down arrow
UP	Up arrow
S&C SHIFT	-CTRL
C	CTRL
S SHIFT	FT
ESC	CAPE

```
   Ix0 |x1 Ix2 |0 Ix4 Ix5 Ix6 |x7 |x8 Ix9 IxA Ix8 IxC IxD IxE IxF |

0x | @| A | B | C | D | E | F | G 1 H 1 1 | J 1 K | L |RETI      N | 0 |
   I C I C I C I C i C I C I C I C I C I C I C I C I I C I C I

1x 1 P | Q | R | S 1 T I U | V 1 W | X | Y | Z | [ | \ | 1 | 1 _ |
   I C I C I C I C I C I C I C I C I C I C I C I C I C I C i C |

2x |SPC|  ! 1 " | q I $ I Z | & | ' I ( | ) I * I + I       I - | . I / I
       I s | S I s I s I S I S I S I S I S | s I S !       I   I   I   I

3x | 0 | 1 1 2 | 3 1 4 | 5 | 6 | 7 | 8 1 9 1 : 1 ; | < | = 1 > | ? I
   I   I   I   I   I   I   i   I   I   I   I   I   I S I S I S I s I

4x|@1 A| B| C1 D| E| F | G I H | I | J | K | L | M | N | 0 |
   !S+LIS+LIS+LIS+LIS+LIS+LIS+LIS+LIS+L!S+LIS+LIS+LIS+LIS+LIS+LIS+LI

5x | P | Q | R | S | T | U | V 1 W | X | Y | Z | C 1 \ I I I 1 _ I
   !S+LIS+LIS+LIS+LIS+LIS+LIS+LIS+LIS+LIS+LIS+LI       I   I   I   I   I

6x | ' 1 a | b 1 c | d | e 1 f I g I h I i I     j  | k | L 1 m 1 n | o 1
   I S 1   I   I   I   I   I   I   I   I   I   I   I   I   I   1   I   I

7x | p I q | r | s I t | u 1 v I w | x | y 1 z | ( | I | ) 1 _ !DELI
   I   I   !   I   !   I   I   !   !   !   I   ! S I S I S I S !    I

8x If0 If1 If2 If3 If4 If5 If6 If7 If8 If9 !TABICPYILFTIRHTIDWNIUP !
   I   !   I   !   I   I   I   !   I   I   !   I   I   I   I   I   I

9x If0 If1 If2 If3 If4 If5 If6 If7 If8 If9 ITABICPYILFTIRHTIDWNIUP !
   ! S ! S I S ! S ! S I S ! S I S I S I S ! s I S I S I S ! S I S I

Ax If0 If1 If2 If3 If4 If5 If6 If7 If8 If9 !TABICPYILFTIRHTIDWNIUP I
   ! C I C ! C I C I C ! C ! C 1 C I C ! C I C I C ! C I C I C ! C I

Bx If0 If1 If2 If3 If4 If5 If6 If7 If8 If9 !TABICPYILFTIRHTIDWNIUP !
   IS&C!S&CIS&CIS&C!S&CIS&CIS&C!S&CIS&CIS&CIS&CIS&CIS&CIS&CIS&CIS&CI

Cx ! M !RETIRETIRETIESCIESCIESCIESCI 0 |DELIDELIDELI @       I SPCISPCISPCI
   I C I S ! C IS&CI     ! S ! C IS&CI C ! S ! C IS&CI s ! S ! C IS&CI

   Dx| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : I   ;  | ,| =1 .|?!
   IS+C!S+CIS+CIS+CIS+CIS+CIS+CIS+CIS+CIS+CIS+CIS+CIS+CIS+CIS+CIS+CI

Ex | @I A | 8 1 C | D | E | F 1 G | H | 1 | J I K | L I M 1 N | 0 |
   IS&CIS&C!S&CIS&CIS&C!S&C!S&CIS&CIS&CIS&CIS&CIS&CIS&CIS&CIS&CIS&CI

Fx | P | Q | R 1 S | T | U | V I W I X | Y | Z I C I \ | 1 1 I _ 1
   !S&C!S&CIS&C!S&CIS&CIS&C!S&C!S&CIS&CIS&CIS&C!S&CIS&CIS&CIS&CIS&CI
```

Appendix B

Bibliography

Panos Programmer's Reference Manual
Acorn Computers Ltd
Part Number 0410,012
Issue 1 1985

Cambridge Co-Processor User Guide
Acorn Computers Ltd
Part Number 0410,000
Issue 1 1985

BBC Microcomputer User Guide
British Broadcasting Corporation
1982

The Advanced User Guide for the BBC Micro
Bray, Dickens and Holmes
Cambridge Microcomputer Centre
1983

Disc Filing System User Guide
Acorn Computers Ltd
Part Number 0403,700
Issue 2 1983

Econet Level 2 File Server User Guide
Acorn Computers Ltd
Part Number 0412,018
Issue 1 1983

Winchester Disc Filing System User Guide
Acorn Computers Ltd
Part Number 0427,000
Issue 1 1984

BBC Microcomputer User Guide
Acorn Computers Ltd
Part Number 0433,000
Issue 1 1984

Index

Acorn Computers Limited
Scientific Division
Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN
Telephone (0223 245200)